VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

MOBILNÍ APLIKACE PRO VYHLEDÁVÁNÍ A SPRÁVU RECEPTŮ

MOBILE APPLICATION FOR RECOMMENDING AND MANAGING COOKING RECIPES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **ANDREJ LONČÍK**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. FRANTIŠEK ZBOŘIL, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Lončík Andrej**
Program: Informační technologie
Název: **Mobilní aplikace pro vyhledávání a správu receptů**
Mobile Application for Recommending and Managing Cooking Recipes
Kategorie: Umělá inteligence

Zadání:

1. Prostudujte metody počítačového vidění a implementaci systémů v prostředí Android.
2. Navrhněte systém, který umožní specifikovat množinu dostupných, případně nedostupných nebo nechtěných surovin. Na základě těch, společně se specifikací omezení, jako jsou například maximální množství kalorií, zdravotní omezení uživatele a podobně, systém navrhne pokrmy a recepty na jejich přípravu. Dále systém umožní recepty spravovat, tzn. uchovávat a to i v různých verzích, kategorizovat, případně hodnotit.
3. Pro získávání popisu dostupných surovin zvažte použití metod počítačového vidění a zpracování hlasového vstupu. Alespoň jedna forma byť jen podpůrná by měla být součástí aplikace. Pokud ne, argumentujte, proč toto není vhodné.
4. Implementujte navržený systém. Ohled berte na snadné použití uživateli a také na zachování jejich soukromí.
5. Navržený systém otestujte na vhodné skupině uživatelů, která bude zahrnovat osoby různého věku a znalosti práci s počítačovými aplikacemi.

Literatura:

- Karthikeyan NG: Machine learning projects for mobile applications : build Android and iOS applications using TensorFlow Lite and Core ML, Birmingham, UK : Packt Publishing, 2018
- Yun Cheng, Aldo Olivares Domínguez: Advanced Android App Architecture (First Edition): Real-world app architecture in Kotlin 1.3, Razeware LLC, 2019

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Zbořil František, doc. Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 11. listopadu 2020

Abstrakt

Cieľom tejto práce je vytvoriť mobilnú aplikáciu pre zariadenia s operačným systémom Android, ktorá slúži pre vyhľadávanie a spravovanie receptov a tvorbu jedálnička. Aplikácia by mala umožňovať hlasový vstup a rozpoznávanie ingrediencií z obrazu. Text práce popisuje celý proces tvorby mobilnej aplikácie od prvotnej myšlienky, cez analýzu konkurencie, návrhu užívateľského rozhrania, implementácie až po zaverečné testovanie a finálne publikovanie. Výsledné riešenie navyše umožňuje vytvárať vlastné recepty aj vyhladávať recepty na základe dostupných ingrediencií. Tie je možné zadať textom, hlasovým vstupom alebo pomocou rozpoznávania z obrazu vďaka nástroju Firebase ML Kit Image Labelling. Súčasťou je autentizácia pomocou Google účtu a uchovávanie užívateľského obsahu vo Firebase Realtime Database. Mobilná aplikácia je zverejnená v obchode Google Play pod názvom **Recipio**.

Abstract

The goal of the submitted thesis is the creation of mobile application for devices using the Android operation system. The main purpose of the application is the discovery and administration of food recipes and meal planning. The functions of the application include voice control and search by a photo or an image. This work describes the whole process of app-creation, beginning from the original idea, followed by the competition analysis, draft of the user interface, its implementation and concluding with the testing and final publication to the Google Play. In addition, the final version of the application offers the feature of creating new recipes or searching for already published ones on the internet based on the ingredients the user possesses. The ingredients can be written in, entered by the user's voice, or recognized from an uploaded image. The photo and image recognition is provided by the Firebase ML Kit Image Labeling tool. Thanks to the Google account authentization, the application is also able to save the user's content in Firebase Realtime Database. Mobile application is published on the Google Play store and is officially named **Recipio**.

Klíčová slova

Android, Firebase, mobilná aplikácia, Obchod Play, rozpoznávanie obrazu, Kotlin, recepty, ingrediencie, nutričné hodnoty, užívateľské rozhranie, Figma, rest API

Keywords

Android, Firebase, mobile application, Google Play, image recognition, Kotlin, recipes, ingredients, nutritions, user interface, Figma, rest API

Citace

LONČÍK, Andrej. *Mobilní aplikace pro vyhledávání a správu receptů*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. František Zbořil, Ph.D.

Mobilní aplikace pro vyhledávání a správu receptů

Prohlášení

Prehlasujem, že som túto bakalárksu prácu vypracoval samostatne pod vedením pana doc. Ing. Františka Zbořila, Ph.D. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Andrej Lončík
10. května 2021

Poděkování

Rád by som poďakoval pánovi doc. Ing. Fratiškovi Zbořilovi za vedenie práce, odbornú pomoc a vecné pripomienky, ktoré mi pomohli pri tvorbe tejto bakalárskej práce. Taktiež by som rád poďakoval všetkým, ktorí sa zapojili do testovania aplikácia a za ich spätnú väzbu.

Obsah

Kapitola 1

Úvod

Mobilné aplikácie už dnes používame pri každodenných činnostiach, bez toho aby sme si to vôbec uvedomovali. Sú navrhnuté tak precízne, že ani nepotrebujeme premýšľať nad tým, ako ich ovládať. Pomáhajú nám eliminovať absurdné situácie, uľahčujú život, nahrádzajú desiatky nástrojov, vytvárajú závislosť. Hlavnou myšlienkou tejto práce je vytvorenie mobilnej aplikácie pre vyhľadávanie a spravovanie receptov. Aplikácia, ktorá nahrádza pôvodné kuchárky a zjednocuje neusporiadané kolekcie zaujmávých receptov. Ja osobne, mám uložené recepty všade možne, len nie po ruke. Na sociálnych sieťach, v poznámkach starého mobilu, na usb kľúči. To je pre mňa jednou z hlavných motivácií pustiť sa do vlastného riešenia mobilnej aplikácie. Dnes už sa dá samozrejme očakávať, že existuje niekoľko fungujúcich a zabehnutých aplikácií, ktoré by túto funkčnosť spĺňali. Problém však nastáva, ak si chcem ako používateľ vytvoriť a uložiť vlastný recept. Nehovoriac o tom, že mnoho základných funkcionalitách je obmedzených kvôli bezplatnému účtu.

Cieľom tejto práce je vytvoriť mobilnú aplikáciu pre vyhľadávanie a spravovanie receptov na platformu Android. Ďalšími funcionalitami aplikácie je tvorba vlastného jedálnička a umožniť vyhľadávanie receptov na základe dostupných ingrediencií. Aplikácia by mala podporovať hlasový vstup a rozpozonávanie ingrediencií z obrazu. Taktiež je potrebné brať ohľad na používateľove intolerancie, diéty a možnosť filtrovania výsledkov podľa nutričných hodnôt. V neposlednom rade je dôležité súkromie a ochrana užívateľských dát. Výsledná aplikácia nesie názov Recipio a je zverejnená v obchode Google Play[1].

Práca je členená do nasledujúcih častí. V kapitole 2 sa venujem analýze konkurenčných aplikácií a existujúcich riešení. V ďalšej kapitole 3 rozoberám všeobecne vývoj mobilných aplikácií a opisujem operačný systém Android z hľadiska obecného fungovania, aj z pohľadu vývojárského prístupu. V kapitole 4 sa venujem procesu návrhu užívateľského rozhrania aplikácie. Následne v kapitole 5 rozoberám návrh systému a voľbu konkrétnych technológií a služieb. V kapitole 6 opisujem finálnu implementáciu mobilnej aplikácie a uvádzam niekoľko konkrétnych zaujímavých prípadov z rady vlastných riešení. V kapitole 7 je opísaný spôsob a priebeh testovania a publikovania výslednej aplikácie. Na záver, v kapitole 8 vyhodnocujem celkovú prácu, získané skúsenost, navrhujem ďalšie kroky a výhliadky do budúcna pre rozvoj aplikácie Recipio.

[1] Aplikácia Recipio dostupná v obchode Google Play https://play.google.com/store/apps/details?id=com.lonchi.andrej.lonchi_bakalarka

Kapitola 2

Existujúce riešenia

Prvý krok práce spočíval v priezkume už existujúcich aplikácií, ktoré sa nachádzajú na trhu. Skrz hlavnú tému tejto práce, sa dá očakávať, že dnes už existujú mobilné aplikácie zastupujúce funkcie digitálnych kuchárok a jedálníčkov. Analýza konkurencie pomáha nielen odhaliť nedostatky existujúcih riešení, ale taktiež odpozorovať fungujúce a odskúšané praktiky. Ťažko odhadnúť celkový počet aplikácií odpovedajúci tématike, preto som výber zúžil len na pár najzaujímavejších počinov, ktoré sú z môjho pohľadu najrelevantnejšie. V úvahu beriem celkové hodnotenie, počet používateľov, dobu fungovania aplikácie a v neposlednom rade jej dohľadateľnosť v rámci obchodu.

Supercook

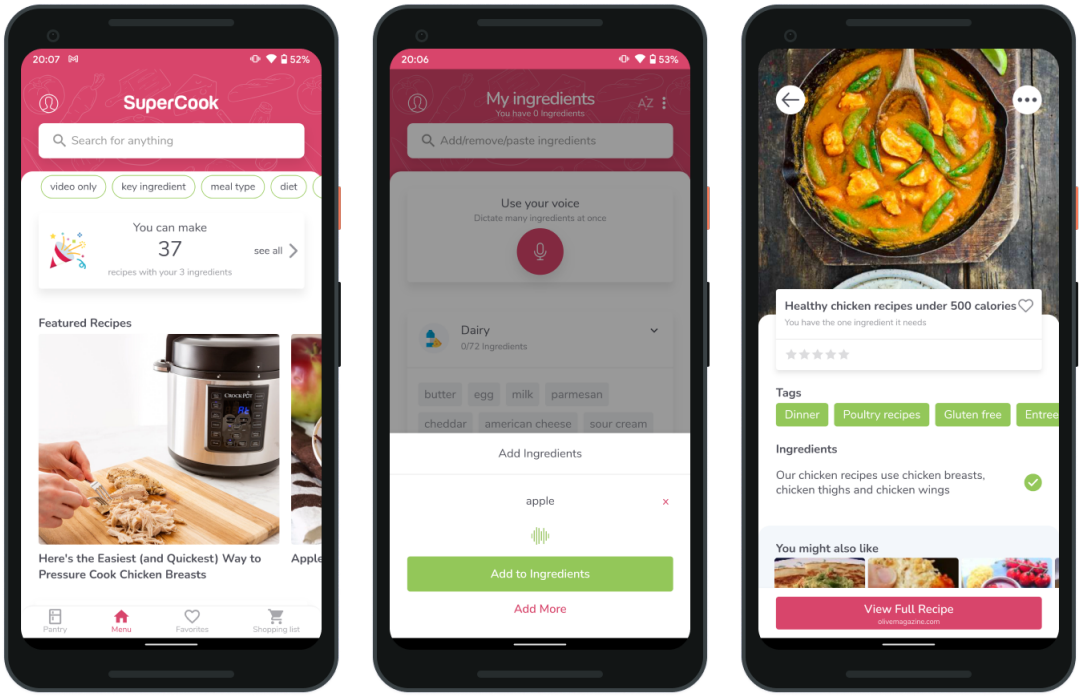
Kľúčovou myšlenkou aplikácie Supercook [1] je vyhľadávanie receptov z dostupných ingrediencií, ktoré má užívateľ "v špajzi". Po vyhľadaní receptov sa suroviny neresetujú, ostávajú v zozname aj do ďalších vyhľadávaní, pokiaľ ich užívateľ neodstráni. Okrem destinácie *Pantry* (špajza), aplikácia obsahuje aj *Shopping list* (nákupný zoznam), ktorý funguje jednoducho ako poznámkový blok v odrážkach. Používateľ si môže recepty pridať do obľúbených. Pridanie ingrediencie do "špajze"podporuje hlasový vstup.

Navigácia aplikácie je riešená pomocou *bottom navigation menu*, samotné destinácie mi však prídu neintuitívne, na prvé použitie až mätúce. Aplikácia pri bežnom používaní nemá plynulý chod a je voľným okom viditeľné, že vykreslenie niektorých snímkov je vynechané. To môže byť spôsobené práve množstvom obrázkom alebo samotnou implementáciou užívateľského rozhrania.

Výsledky vyhľadávaných receptov sú rozdelených do niekoľkých rôznych kategórií, ako napríklad *10 ingredient or less*, *Gluten free recipes*, *5 ingredient or less* a iné. Detail samotného receptu obsahuje nutričné hodnoty a ingrediencie. Pre zvyšok informácií, ako je napríklad postup, je potrebné kliknúť na tlačítko, ktoré používateľa presmeruje na webovú stránku konkrétneho receptu. Diéty a intolerancie sú zahrnuté v *tagoch* receptu, ale nevplývajú na samotné vyhľadávanie.

Aplikácia je bezplatná a má viac než 500 000 stiahnutí s priemerným hodnotením 4.6 hviezdičiek.

[1] Supercook https://play.google.com/store/apps/details?id=com.supercook.app&hl=cs&gl=US



Obrázek 2.1: Ukážka užívateľského rozhrania aplikácie Supercook - nájdené recepty, hlasový vstup, detail receptu

Yummly

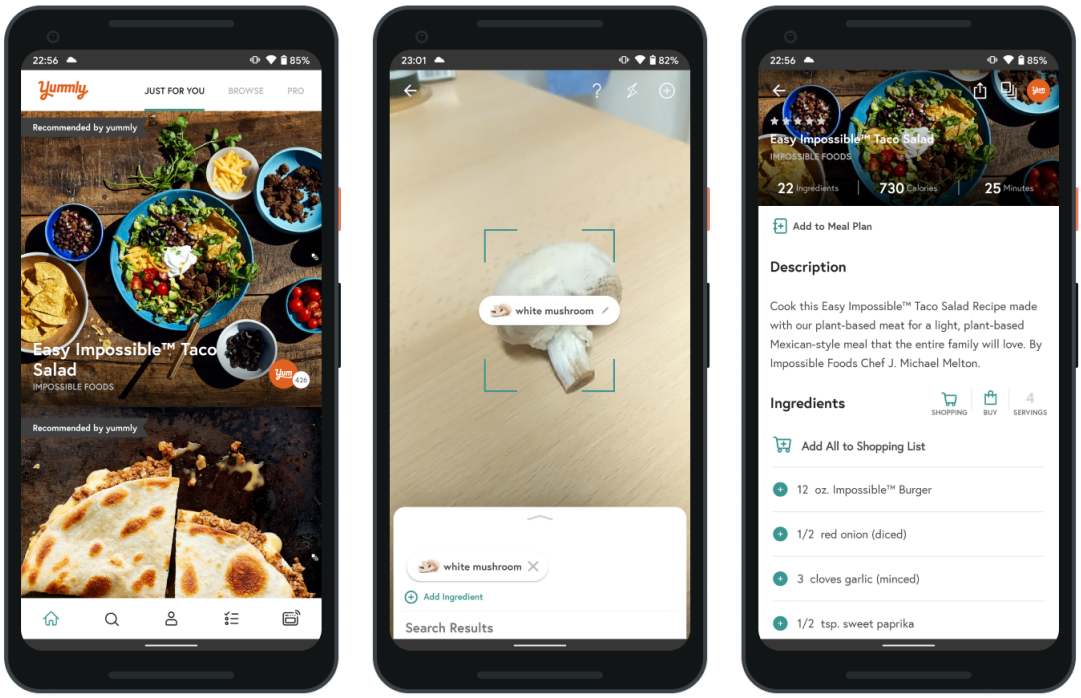
Aplikácia Yummly[2] je najpopulárnejšia z tohto výberu a funguje už od roku 2009. K dnešnému dňu má viac než 5 miliónov stiahnutí s obdivuhodným hodnotením 4.5 hviezdičiek. Užívateľské rozhranie je veľmi prepracované, intuitívne a zvlášť dobre implementované, keďže aj napriek množstvu obrázkov s vysokým rozlíšením a mierne neštandardnými komponentami aplikácia funguje plynule. Pred prvým spustením sa používateľ musí registrovať respektíve prihlásiť, podporované je prihlásenie cez sociálne siete Google a Facebook. Nasleduje *onboarding*, kde si používateľ zvolí obľúbenú kuchyňu, alergény, diéty, neobľúbené jedlá a svoje kulinárske schopnosti. Po nastavení preferencií aplikácia okamžite ponúkne prechod na premium plán. Cena mesačného premium plánu je 140 CZK.

Navigácia je taktiež riešená pomocou bottom navigation menu, ktoré obsahujke 5 destinácií. Prvá destinácia, domov, odporúča používateľovi personalizované recepty. Aplikácia je evidentne podporovaná rozsiahlim back-endom. Druhá destinácia slúži pre vyhľadávanie receptov. V profile si používateľ môže prezerať obľúbené a uložené recepty, tie sa dajú ukladať do rôznych kolekcií. Menu obsahuje aj destináciu pre plánovanie jedálnička, ale jedná sa o premium funkcionalitu. Posledná destinácia nesie názov *Smart Thermometer*, ale jedná sa opäť o premium funkcionalitu.

Vyhľadavánie receptov sa dá filtrovať a zahŕňa predvolené diéty a alergény. Samotné vyhľadávanie receptov nepodporuje hlasový vstup. Recepty je možné vyhladať aj zadaním ingrediencií, tento proces je však tak trochu nešťastne schovaný za ikonou fotoaparátu. Pridanie ingrediencie prebieha primárne real-time rozoznávaním obrazu s využitím fotoaparátu. Rozpoznávanie funguje veľmi dobre a ukážku je možné vidieť na obrázku 2.2. Ingre-

[2]Yummly https://play.google.com/store/apps/details?id=com.yummly.android&hl=cs&gl=US

diencia sa dá pridať aj manuálne zadaním vstupného reťazca. Detail receptu je prehľadný a obsahuje všetky dôležité informácie vrátane ingrediencií, postupu a nutričných hodnôt. Za najväčšiu nevýhodu aplikácie považujem obmedzenú funkcionalitu bezplatnej verzie a časté navádzanie používateľa na premium plán.



Obrázek 2.2: Ukážka užívateľského rozhrania aplikácie Yummly - odporúčané recepty, real-time rozpoznávanie ingrediencií z obrazu, detail receptu

SideChef

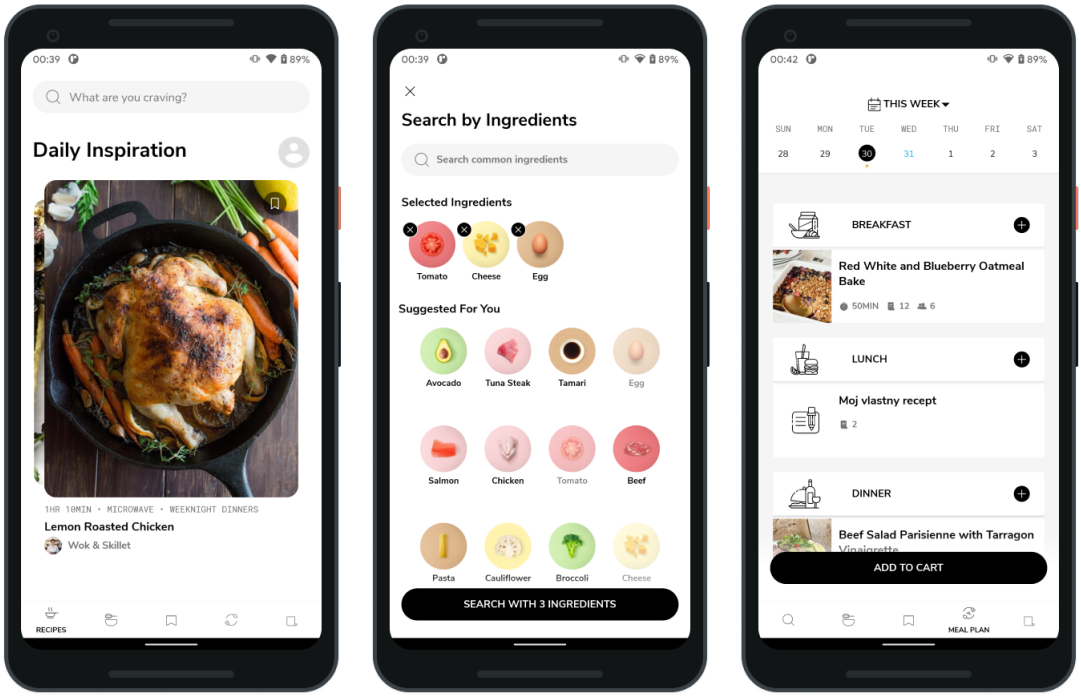
Aplikácia SideChef[3] má viac než 500 000 stiahnutí a hodnotenie 4.5 hviezdičiek. Po spustení je nutné sa prihlásiť, prihlasovanie podporuje sociálne siete. Užívateľské rozhranie je veľmi moderné a čisté vďaka farebnej kombinácií čiernej a bielej. Navigácia je riešená pomocou *bottom navigation menu* a obsahuje mini interakcie, ktoré vylepšujú užívateľskú skúsenosť. Menu je tvorené z piatich položiek: *recipes*, *explore*, *saved*, *meal plan* a *grocery* (recepty, objaviť, záložky, jedálniček a zoznam potravín).

Recepty sú doporučované veľmi zaujímavým spôsobom – viz obrázok 2.3. Recept sa dá uložiť do záložiek, tie fungujú ako kolekcie, ktoré si môže používateľ spravovať. Detail receptu obsahuje všetky potrebné informácie (ingrediencie, inštrukcie, nutričné hodnoty, podobné recepty). Ku niektorým receptom je k dispozícií aj videonávod. Zaujmavá je taktiež funkcionalita *step by step mode*, ktorá spustí komentár k jednotlivým inštrukciám. Okrem receptov sa v aplikácií nachádzajú aj blogové články. Vlastný recept nie je možné vytvoriť.

Recepty je možné vyhľadať vstupným reťazcom alebo podľa ingredencií. Vyhľadávanie ingrediencií je zvládnuté veľmi dobre (ukážka na obrázku 2.3). Hlasový vstup ani rozoznávanie ingredencií z obrazu nie je k dispozícií. Niektoré recepty sú uzamknuté, nakoľko sú dostupné len pre *premium* používateľov. Premium plán stojí 150 CZK mesačne.

[3]SideChef urlhttps://play.google.com/store/apps/details?id=com.sidechef.sidechef

Jedálniček nemá časové obmedzenia a je dostupný aj v bezplatnej verzií aplikácie, čo považujem za veľké plus. Staticky sa skladá z troch položiek: *breakfast*, *lunch* a *dinner* (raňajky, obed a večera). Pridávanie receptu však nie je príliš užívateľsky príjemné, po pridaní receptu do jedálnička sa proces neukončí automaticky, užívateľ sa musí preklikať naspäť cez 4 obrazovky.



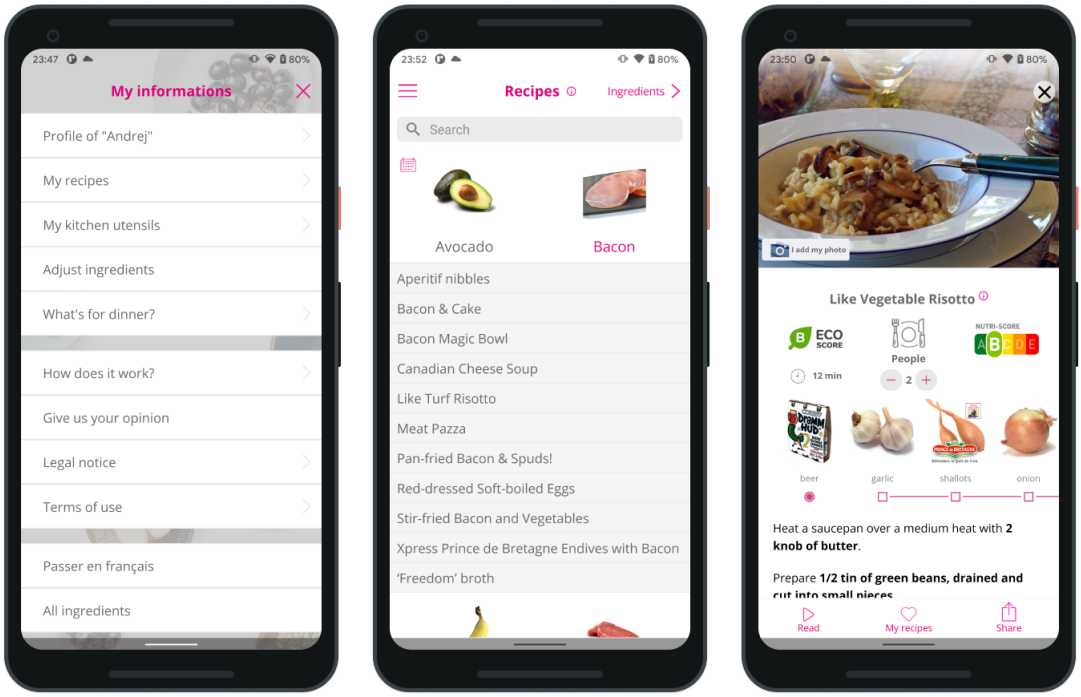
Obrázek 2.3: Ukážka užívateľského rozhrania aplikácie SideChef - odporúčané recepty, vyhľádavanie podľa ingrediencií, jedálniček

Magic Fridge

Posledná aplikácia sa prezentuje jednoduchými receptami a snaží sa zabrániť zbytočnemú plýtvaniu jedlom. Magic Fridge[4] má v Obchode Play viac než milión stiahnutí a hodnotenie 4.1 hviezdičiek. Pri prvom spustení sa používateľ musí prihlásiť, opäť je podporované aj prihlásenie cez sociálne siete Google a Facebook. Užívateľské rozhranie mi príde zastaralé a neintuitívne. Navigáciu rieši bočné menu doplnené o nelogické animácie. Niekedy sa menu zobrazí na obrazovke príchodom spravej strany, inokedy zasa zhora, prípadne zdola. Navyše je samotné menu neprehľadné a neindikuje aktuálnu pozíciu používateľa – viz. obrázok 2.4.

Recepty je možné vyhľadať iba cez ingrediencie. Neprišiel som na to, akým spôsobom skombinovať rôzne ingrediencie. Hlasový vstup nie je podporovaný, rovnako tak ani rozoznávanie ingrediencií z obrazu. Detail receptu zobrazuje potrebné ingrediencie aj postup prípravy. Každý recept je hodnotený ekologickým a nutričným skóre. Po kliknutí na nutričné skóre sa zobrazia aj konkrétne nutričné hodnoty, odhalil som to však náhodným kliknutím. Recept je možné pridať do obľúbených. Aplikácia navyše obsahuje rôzne formy reklamy, vyskakovacie pop-up okná a aj reklamné banery na konkrétne produkty.

[4]Magic Fridge https://play.google.com/store/apps/details?id=fr.haruni.frigomagic&hl=cs&gl=US



Obrázek 2.4: Ukážka užívateľského rozhrania aplikácie Magic Fridge - menu, vyhľadávanie receptov, detail receptu

Vyhodnotenie

Osobne sa mi najviac páčila aplikácia Yummly, hlavne vďaka rýchlemu, svižnému užívateľskému rozhraniu a podpory pre rozpoznávanie surovín z obrazu. Väčšina aplikácií riešila navigáciu **spodným navigačným menu**, čo považujem za najlepšiu možnosť. Prihlásenie cez **sociálne siete** sa mi užívateľsky javí veľmi prívetivé a rád by som ho zahrnul do aplikácie. Pri zadávaní vstupného reťazca by som upriamil väčšiu pozornosť na hlasový vstup a rovnako tak aj zvýraznil možnosť zadať potravinu pomocou rozpoznávania obrazu. Ani v jednej aplikácií nebola **možnosť vytvoriť si vlastný recept**. Proces tvorby vlastného receptu je jednou z kľúčových vecí, ktorou by mala disponovať výsledná aplikácia .

Kapitola 3

Vývoj mobilných aplikácií pre Android

V tejto kapitole sa zameriam na operačný systém Android. Priblížím jeho architektúru, históriu, aktuálne verzie a základné princípy vývoja. Ešte pred tým však zmienim spôsoby vývoja mobilných aplikácií obecne a vysvetlím rozdiely medzi natívnym, a čoraz viac populárnejším, cross-platformovým vývojom.

3.1 Natívny vs. cross-platformový vývoj

Natívny prístup predstavuje oddelený vývoj pre obe platformy. Zvlášť prebieha vývoj na Android a zvlášť prebieha vývoj na iOS. Pri cross-platformovom prístupe naopak prebieha len jeden vývoj, ktorého výstupom je aplikácia kompatibilná s viacerými platformami. Na prvý pohľad sa tak môže javiť cross-platform vývoj ako výhodnejší. Problematika je však omnoho hlbšia a neexistuje univerzálna odpoveď.

Oba prístupy majú svoju radu výhod a nevýhod. Dôležité je mať vopred stanovené funkcionality plánovanej aplikácie, z ktorých jasne vyplýva k čomu všetkému je potrebné mať prístup a ako moc do hĺbky operačného systému bude potrebné zájsť pri samotnej implementácií. Predsa len je rozdiel, ak jedna aplikácia slúži len ako zobrazovač obsahu na pár obrazovkách – viz Wikipedia[1] – a druhá aplikácia, ktorá sa pripája cez bluetooth k iným zariadeniam, prehľadáva súbory či aktualizuje firmware pripojeným zariadeniam – viz nRF Connect[2].

Natívny vývoj

Exkluzívny vývoj pre každú platformu zvlášť zaberie viac času a zvýši náklady. Vyžaduje minimálne jedného Android developera a jedného iOS developera, ktorí musia implementovať celú aplikáciu od piky. Opominutie jednej platformy znamená stratu potencionálnych užívateľov. Na druhú stranu s natívnym vývojom sa dá dosiahnuť vyššia výkonnosť a optimalizácia aplikácie. Taktiež väčšia kontrola a hlbšia interakcia s operačným systémom, bezproblémová používateľská skúsenosť a takmer žiadne obmedzenia, ak opominieme hardwarové limity. Natívny mobilný vývoj má zastúpenie v obrovskej vývojárskej komunite už viac než desať rokov. Vďaka tomu sú dohľadateľné riešenia aj na tie najšpecifickejšie

[1] Wikipedia na Google Play https://play.google.com/store/apps/details?id=org.wikipedia&hl=en

[2] nRF Connect for Mobile na Google Play https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp&hl=en&gl=US

problémy. Pri nových funkciách a obmedzeniach, ktoré prichádzajú s novými verziami operačných systémov, je automaticky poskytovaná technická dokumentácia, záruka správnej funkčnosti a prípadná podpora pre hlásenie problémov a chýb.

Natívny vývoj pre Android a pre iOS je odlišný hneď v niekoľkých veciach. Obe platformy majú svoje vlastné vývojové prostredie – Android Studio pre vývoj Android aplikácií – Xcode pre vývoj iOS aplikácií. Xcode funguje iba na operačnom systéme macOS, kdežto Android Studio je možné nainštalovať aj na Windowse, macOS i Linuxe. Z toho vyplýva ďalší podstatný rozdiel, Android je omnoho viac otvorenejší a prispôsobiteľnejší systém, kdežto iOS je viac uzamknutejším a bezpečnejším systémom. Má určené hranice, ktoré jasne definujú a zjednocujú chovanie systému. Dokonale tak spadá do Apple ekosystému. To je jeden z hlavných dôvodov, prečo vývoj iOS aplikácie trvá kratšie než vývoj rovnakej aplikácie na Android [6]. Ďalším z dôvodov je *device fragmentation*[3]. Zatiaľ čo Apple vydalo len niekoľko generácií zariadení iPhone, zariadení s operačným systémom Android sú tisíce. Zariadenia majú rôznu fyzickú veľkosť obrazovky, rozlíšenie a prípadnú nadstavbu operačného systému od výrobcu. Aplikácie sú publikované na Google Play resp. App Store a politika schvalovania aplikácií je taktiež rozdielna. Spoločne však platí, že schvalovací proces aplikácií je z roka na rok prísnejší.

Cross-platformový vývoj

Záujem o cross-platformový vývoj v posledných rokoch narastá. Na prvý pohľad sa môže zdať omnoho výhodnejšie zvoliť cross-platformový prístup, nakoľko sa vyvinie jedna aplikácia, ktorá sa následne vyexportuje pre Android a iOS platformu. Logicky sa ušetrí čas aj náklady, na vývoj postačí jeden developer. Prípadné zmeny a aktualizácie sa opäť nemusia vykonávať dvojmo, kód je lepšie udržovateľný a nevznikajú nechcené rozdiely medzi Android a iOS verziou. Tento prístup však prináša aj istú radu nevýhod. Spravidla sa totižto jedná len o cross-platformové frameworky, ktoré len fungujú ako vrstva nad samotným operačným systémom. Z toho vyplýva istá rada nevýhod a obmedzení, či už sa jedná o výkon či prístup k periférium zariadenia. Stručne predstavím aspoň zopár konkrétnych frameworkov slúžiacich na cross-platformový vývoj mobilných aplikácií.

Flutter

Flutter je open-source vývojový nástroj priamo od Google vydaný v máji 2017. Napriek svojmu mladému veku sa veľmi rýchlo zaradil medzi najlepšie a najobľúbenejšie nástroje pre tvorbu cross-platformových aplikácií. Základný princíp Flutteru je v tom, že so všetkými UI elementami pracuje ako s widgetmi. Je založený na jazyku Dart, relatívne novom programovacom jazyku, ktorý zdieľa rovnaké princípy a metodiky ako natívne programovacie jazyky Kotlin a Swift. V marci roku 2021 bol predstavený Flutter 2, ktorý okrem mobilných aplikácií už plne podporuje aj vývoj webových aplikácií[4].

React Native

React Native bol predstavený v roku 2015 spoločnosťou Facebook. Framework je založený na Reacte – JavaScriptová knižnica pre tvorbu vysoko responzívnych používateľských rozhraní. Veľká časť kódu je totožná s JavaScriptom, čo umožňuje jednoduchú adaptáciu pre web developerov. React Native však nie je tak úplne cross-platfoprmovým frameworkom.

[3]Device fragmentation https://developer.android.com/about/dashboards
[4]Flutter https://flutter.dev

Pri použití funkcionalít ako sú kamera alebo akcelometer je potrebné mať separátny kód pre Android a iOS. V posledných rokoch však React Native dopláca na nekonzistentnosť aktualizácií a komplikovanosť debuggovania pri vývoji[5].

Xamarin

Xamarin bol prvým dostupným open-source frameworkom pre cross-platformový mobilný vývoj. Bol uvedený na trh v roku 2011, ale skutočnú dôveryhodnosť si našiel až v roku 2016, kedy ho akvizovala spoločnosť Microsoft. Základom je objektovo orientovaný jazyk C#, kvôli ktorému veľa vývojárov uprednostní skôr Flutter alebo React Native, nakoľko je jednochšie sa ich naučiť. Aplikácie vytvorené pomocou Xamarinu však fungujú ako natívne aplikácie a to na rôznych platformách vrátane Androidu, iOSu, tvIS, watchOS, macOS a Windows[6].

Apache Cordova

Framework Cordova vznikol už v roku 2009 pod menom PhoneGap. Neskôr v roku 2011 bol rebrandovaný a poskytnutý organizácií Apache, na základe čoho vznikol aktuálny názov. Rovnako ako React Native, je aj Apache Cordova lákavou možnosťou pre web developerov, ktorí by svoje poznatky a skúsenosti z HTML, CSS a Java Scriptu chceli uplatniť pri tvorbe mobilnej aplikácie pre Android a iOS. Navyše je dostupných mnoho nástrojov a knižníc, napríklad **Ionic** pre prácu s UI a integráciu doplnkových služieb. Najväčším problémom je však vykreslenie kódu pomocou WebView, čo má za následok výraznú stratu výkonu. Cordova absolútne nie je vhodným frameworkom pre vývoj stredných či veľkých mobilných aplikácií[7].

Kotlin Multiplatform

Najnovším prírastkom medzi cross-platformový vývoj je Kotlin Multiplatform, ktorý naberá na popularite najmä u Android Developerov, keďže jeho základom je jazyk Kotlin. Umožňuje developerom oddeliť UI od back-endu mobilnej aplikácie. Práve pod týmto back-endom sa myslí bussines logika a spoločný prenositeľný kód pre obe mobilné verzie. Zdieľaný kód je potom doprevádzaný špecifickými Android respektíve iOS APIs a UI komponentami prebraných z natívneho mobilného vývoja[8].

[5] React Native https://reactnative.dev
[6] Xamarin https://visualstudio.microsoft.com/cs/xamarin/
[7] Apache Cordova https://cordova.apache.org
[8] Kotlin Multiplatform https://kotlinlang.org/lp/mobile/

3.2 Operačný systém Android

Operačný systém Android je open-source platforma založená na báze Linuxu, zameraná pôvodne na mobilné zariadenia. Jeho vývoj spadá pod technologického giganta – Google. Aktuálne je Android najrozšírenejším mobilným operačným systémom [2]. Dnes už je systém multiplatformový a používa sa nielen v mobilných zariadeniach, ale aj v televíziach, tabletoch, chytrých hodinkách či dokonca autách.

Najväčšou výhodou a zároveň slabinou operačného systému Android je jeho otvorenosť, či už zo strany výrobcov, developerov alebo koncových užívateľov. Vďaka tejto otvorenosti používa Android vo svojich zariadeniach obrovská rada výrobcov mobilných zariadení, na základe čoho je systém rozšírený do celého sveta. Systém však nie je možné optimalizovať pre tak širokú škálu hardwarových konfigurácií. Mnohí výrobcovia si tak často ohýbajú systém podľa vlastných potrieb, upravujú konfiguráciu a implementujú vlastné nadstavby operačného systému – viz MIUI[9] od Xiaomi alebo One UI[10] od Samsungu. To prináša mnoho komplikácií pri vývoji mobilných aplikácií, pretože výsledné chovanie môže byť na finálnych zariadeniach rozdielne, prípadne úplne nefunkčné.

Verzie Androidu

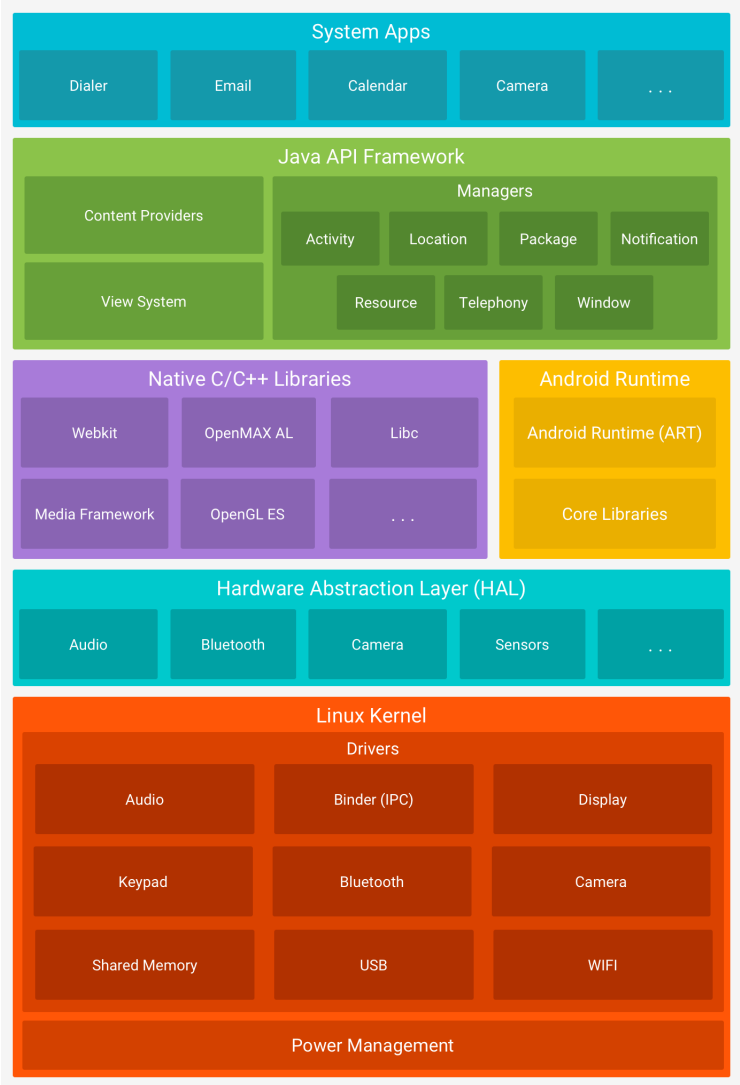
Google každoročne organizuje konferenciu Google I/O[11], ktorá prebieha spravidla v máji a predstavujú na nej novú verziu operačného systému Android. Aktuálna verzia Android 11 bude tak už o pár mesiacov nahradená novou verziou – Android 12. Zaujímavosťou je, že v minulosti Google ku číslu verzie spájal aj meno verzie, ktorá niesla názov po nejakej cukrovinke – Android KitKat, Lillipop, Marshmallow, Oreo a iné. Neskôr Google od tejto tradície upustil a ostal iba pri číslovaní verzie, keď v roku 2019 vydal Android 10. Nové verzie Androidu však nemajú takú dostupnosť ako konkurenčný systém iOS. Aktualizácie sú k dispozícii priemerne 1 až 2 roky od vydania smartphonu, dôvodom sú spomínané nadstavby operačného systému od rôznych výrobcov mobilných zariadení.

Architektúra

Architektúra systému Android sa skladá z piatich vrstiev zobrazených na obrázku 3.1.

- **Systémové aplikácie -** najvyššia vsrtva architektúry, ktorú tvoria systémové aplikácie dostupné pre koncového používateľa. Patrí medzi ne napríklad fotoaparát, prehrávač multimédií, kalendár a iné. Ďalšie aplikácie je možné doinštalovať.
- **Java API framework -** sprostredkuváva aplikáciam prístup k hardwarovým zdrojom zariadenia. Stará sa jednoduché opakované využitie existujúcich komponent s ohľadom na bezpečnosť jednotlivých aplikácií. Figuruje taktiež ako poskytovateľ obsahu, správca zdrojov a aktivít.
- **Natívne C/C++ knižnice -** systém Android poskytuje celú radu knižníc, ktoré sú prístupné skrz aplikačný framework a sú pôvodne napísané a prebrané z jazyka C respektíve C++. Patria ku nim:
  - **OpenGL ES -** grafická knižnica pre 3D

[9] MIUI https://en.miui.com
[10] One UI https://www.samsung.com/us/apps/one-ui/
[11] Google I/O https://events.google.com/io/



Obrázek 3.1: Architektúra systému Android, prebrané z [3]

  - **SGL -** knižnica základého zobrazenia 2D
  - **Media Framework -** knižnica pre prácu s multimédiami
  - **SQLite -** odľahčená verzia SQL
  - **WebKit -** knižnica pre prácu s internetom

- **Android behové prostredie -** vrstva, ktorá má na starosti beh aplikácií. Každá aplikácia beží vo vlastnom procese.
- **Abstraktná hardwarová vsrtva -** poskytuje štandardné rozhranie aplikačnému frameworku ku špecifickým hardwarovým komponentám ako je napríklad bluetooth a kamera.
- **Linuxové jadro -** najnižšia vrstva architektúry založená na Linuxovom jadre. Spravuje základné systémové služby ako je správa procesov, správa pamäte, práca s ovládačmi hardwarových prvkov a iné.

3.3 Špecifikácie vývoja pre Android

Vývoj aplikácií pre platformu Android prebieha už od roku 2008 a odvtedy prešiel radou zmien a vylepšení. Za zmienku určite stojí **Android Jetpack**[12], kolekcia softwarových komponent, ktorá zavádza štandardy, redukuje kód a zjednodušuje množstvo zložitých úkonov. Android Jetpack je vyvinutý priamo firmou Google a vo vývoji sú stále nové funkcionality. Z môjho pohľadu a vlastných skúseností, Android Jetpack výrazne ovplyvnil vtedajší vývoj, mnohé veci redefinoval a nasmeroval ich správnym smerom.

Vývojové prostredie

Hlavným vývojovým prostredím je **Android Studio**[13], aktuálna verzia 4.1.2. Je vytvorené na báze IntelliJ IDEA od spoločnosti JetBrains a špeciálne orientovaný pre vývoj Android aplikácií pre zariadenia všetkých podporovaných platforiem. V minulosti bol používané aj vývojové prostredie Eclipse. Android Studio ponúka niekoľko nástrojov pre uľahčenie vývoja:

- **Database Inspector -** real-time prehľadávanie obsahu databázy spustenej aplikácie na mobilnom zariadení
- **Emulator -** knižnica zariadení, ktoré sú k dispozícií pre spustenie emulácie
- **Lint check -** analýza kódu, pomáha pri dodržovaní štandardov a dbá o najefektívnejší zápis kódu
- **Profiler -** zobrazuje a meria vyťaženie pamäte RAM, CPU a internetovú komunikáciu

Jazyk

Dlhé roky bol hlavným podporovaným jazykom Java, ale vo februári 2016 Google oznámil akvizíciu jazyka **Kotlin**, vyvinutý firmou JetBrains. Kotlin je statický typovaný jazyk určený pre zariadenia podporujúce Javu. Syntaxou podľa môjho názoru pripomína Javu a Python, podobnosť nájdeme aj u jazyka Swift. Hlavnými výhodami Kotlinu oproti Jave sú:

- **Null Safety -** eliminuje riziko *NullPointerException* [9]
- **Vysoko interoperabilný -** umožňuje vytvárať zmiešané projekty so zdrojovými súbormi v Kotline i Jave
- **Funkčné programovanie -** využíva koncepty z funkčného programovania akými sú napr. lambda výrazy
- **Redukcia kódu**

Aktivita

Aktivita je jedna z najzákladnejších komponent v rámci vývoja Android aplikácie. Prepája užívateľské rozhranie s nadväzujúcou logikou. Zachytáva a spracováva všetky používateľské akcie a aktualizuje UI príslušnými reakciami. Aplikácia začína tým, že sa vytvorí a spustí

[12] Android Jetpack https://developer.android.com/jetpack
[13] Android Studio https://developer.android.com/studio

práve jej prvá aktivita. Aktivita vytvára ďalšie aktivity a môžu spolu komunikovať a predávať si dáta.

Pôvodne v Androide jedna aktivita odpovedala jednej obrazovke, postupom času sa však obsah prenechal fragmentom a počet aktivít by mal byť čo najnižší. Google dokonca odporúča tzv. *Single acitivity prístup* [7], čo je však v praxi celkom zložité realizovať. Každá aktivita má svoj životný cyklus (obrázok 3.2), ktorý sa skladá z niekoľkých stavov. Na jednotlivé stavy je možné reagovať vďaka funkciám:

- **onCreate() -** vytvorení novej inštancie aktivity. Ak aplikácia podporuje *landscape mode*, pri prípadnom pretočení je pôvodná aktivita vytvorená znovu a je volaná táto metóda opäť
- **onStart() -** aktivita je vytvorená a je zobrazená
- **onResume() -** aktivita sa dostala do popredia a začala reagovať na používateľove akcie
- **onPause() -** aktivita prešla do pozadia, je však stále viditeľná. Tento stav nastáva napríklad aj pri zobrazení dialógového okna
- **onStop() -** aktivita už nie je viditeľná
- **onRestart() -** inštancia aktivity stále existuje, nie je však zobrazená, ale o chvíľu už bude opäť viditeľná
- **onDestroy() -** aktivita už nie je naďalej potrebná a bude zničená. Volanie tejto metódy nie je zaručené

Fragment

Fragment je ďalšou veľmi používanou komponentov pri vývoji moderných Android aplikácií podľa odporúčaných postupov. Jedná sa o modulárnu časť aktivity, ktorá ma svoj vlastný životný cyklus (obrázok 3.2). Životný cyklus fragmentu je podobný životnému cyklu aktivity, nadobúda však viac stavov. Fragment existuje len v rámci aktivity a je na nej závislý. Vďaka svojej modulárnosti je znovupoužiteľný a v rámci aktivity môže existovať niekoľko fragmentov.

Služby

Komponenty bežiace na pozadí nezávisle od aktivit či fragmentov. Majú svoj vlastný životný cyklus a zabezpečujú beh dlhodobých procesov. Môžu byť doprevádzané notifikáciou. Príkladom takej služby je navigácia alebo prehrávač hudby.

Layouty

Základné komponenty pre zobrazenie užívateľského rozhrania. Starajú sa o rozloženia a usporiadania UI prvkov a komponent. Existuje viacero typov layoutov. Najvýkonnejšou je Constraint Layout. Vždy však treba prehodnotiť náročnosť požadovaného UI a v ideálnom prípade použiť aj menej náročné layouty na vykreslenie, ako sú napr. LinearLayout, Frame Layout či Relative Layout.

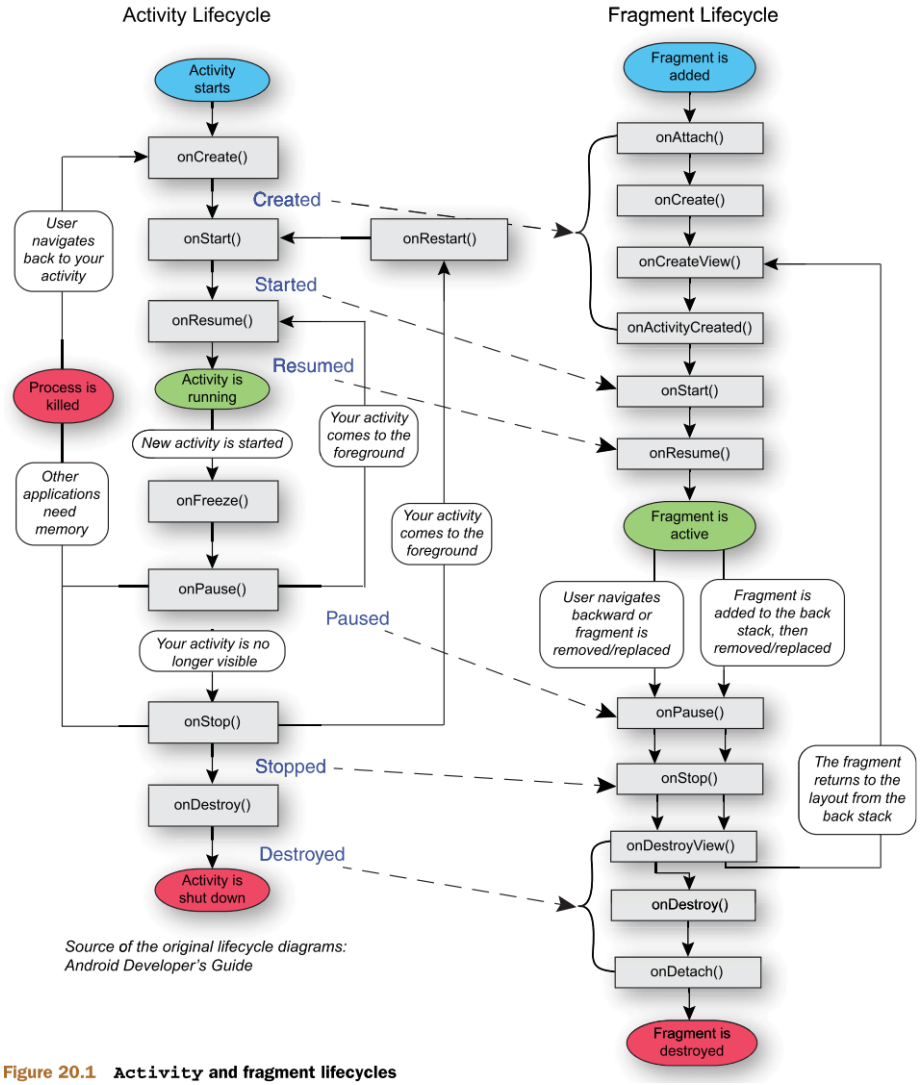


Figure 20.1 **Activity and fragment lifecycles**

Obrázek 3.2: Životný cyklus aktivity a fragmentu, prebrané z [11]

Zdroje

Multimédia, môžu byť rôzneho formátu. Jedná sa o obrázky, vektory, texty, preklady, zvuky, ikony, videá a iné.

Uchovávanie dát

V androide existuje niekoľko spôsobov ako ukladať dáta:

- **Shared preferences -** najjednoduchší spôsob uchovávania dát. Funguje na princípe v ktorom jednému univerzálnemu kľúču prilieha práve jedna hodnota. Uchovávať sa dajú len základné dátové typy. Určené iba pre uchovávanie malého objemu dát, napr. filter zapnutý respektíve vypnutý

- **Databáza -** systém Android vie pracovať s databázou SQLite, ktorá je ideálnym spôsobom ukladania užívateľských informácií, s ktorými aplikácia pravidelne pracuje, napr. dáta o užívateľovi viditeľné v profile
- **Interné úložisko -** chránené a uzamknuté úložisko dostupné iba pre samotnú aplikáciu. Vhodné pre binárne súbory, ktoré nemajú byť verejne dohľadateľné, napr. súbor firmwaru pre bluetooth zariadenia
- **Externé úložisko -** dostupné úložisko v zdielanej pamäti telefónu. Majú k nemu prístup aj ostatné aplikácie s povoleniami aj samotný užívateľ. Vhodné pre verejne dostupné binárne súbory, napr. finálne upravené fotografie z postprodukčnej aplikácie

Gradle

Android projekt sa skladá z množstva zdrojových súborov, rôznych foromátov. Plugin Gradle slúži ako build systém Android Studia. Výstupom je APK[14] súbor, ktorý slúží ako inštalačný súbor mobilných aplikácií na operačnom systéme Android.

[14] APK - Android Package

Kapitola 4

Návrh užívateľského rozhrania

Užívateľské rozhranie je veľmi dôležitou súťasťou mobilných aplikácií. Definuje akým spôsobom bude používateľ aplikáciu ovládať a zodpovedá za celkový dojem, ktorý si používateľ z aplikácie odnesie. Všeobecne platí, že používateľské rozhranie by malo byť hlavne intuitívne. Veľmi dobre to vystihuje citát Martina Leblanca: *"A user interface is like a joke. If you have to explain it, it's not that good."* [8]. V nasledujúcej kapitole preto priblížim odporúčané dizajnové štandardy u mobilných aplikácií, vhodné nástroje pre tvorbu užívateľského rozhrania a samotný priebeh vývoja finálneho užívateľského rozhrania aplikácie.

4.1 Material Design

Material Design je *design system* vytvorený spoločnosťou Google, ktorý má pomôcť vytvárať vysoko kvalitné rozhranie a užívateľskú skúsenosť pre mobilné a webové platfromy. *Design system* je kolekcia znovupoužiteľných komponént, ktorá sa riadi odskúšanými a fungujúcimi normami. Spomínané komponenty sú vývojárom k dispozícií pre jednoduchšiu implementáciu užívateľského rozhrania v rámci finálneho produktu. Google predstavil Material Design v júni roku 2014 a neustále ho vyvíja na základe prichádzajúcih technologických zmien a užívateľských potrieb. Material design pokrýva celú radu potrieb užívateľského rozhrania, vrátane:

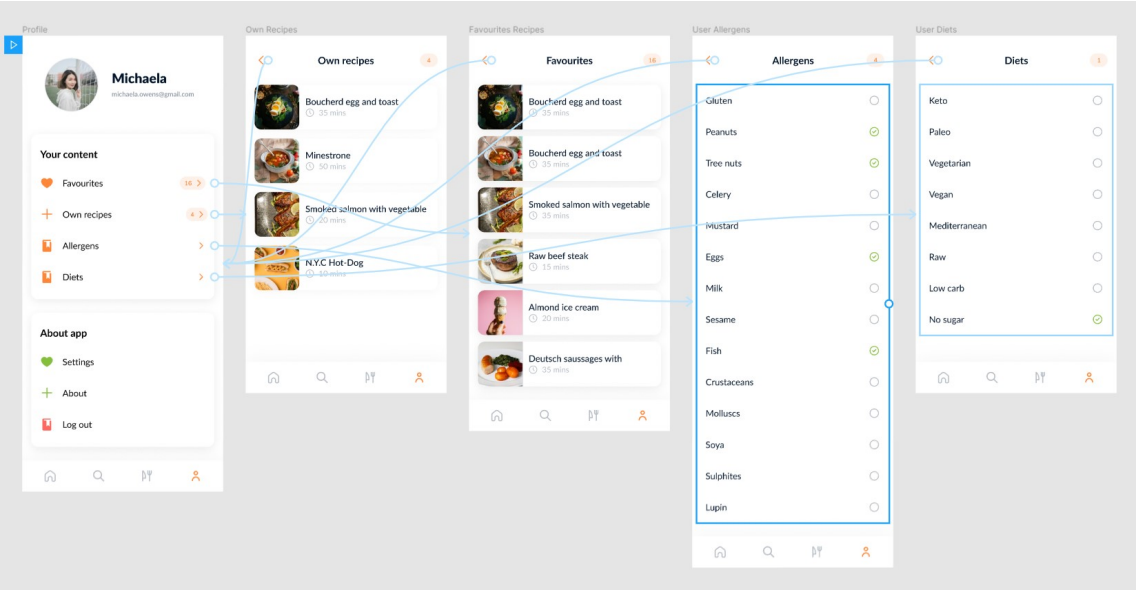
- **Zobrazenie -** umiestnenie a usporiadanie obsahu pomocou komponént, ako sú *cards*, *lists* a *sheets*. Pracuje taktiež s prostredím a využíva hĺbku, povrch a vrhané tiene komponént.
- **Navigácia -** umožňuje užívateľom prirodzene sa pohybovať v rozhraní pomocou komponent ako *navigation drawers*, *bottom navigation menu* a *tabs*.
- **Akcie -** umožňuje užívateľom vykonávať akcie a úlohy využitím komponent ako je napríklad *floating action button*.
- **Vstup -** dbá aby používatelia mohli zadávať vstup alebo uskutočňovať výber vďaka komponentám ako *text inputs*, *chips* a *selection controls*.
- **Komunikácia -** upozorňuje používateľov na kľúčové informácie a správy pomocou komponént, ako sú napráklad *snackbars*, *banners* a *dialogs*.

4.2 Nástoje pre tvorbu UI/UX

S narastajúcim záujmom o vývoj mobilných aplikácií narastá aj počet nástrojov pomáhajúcich pri tvorbe užívateľského rozhrania a používania. Zameriam sa len na tie, ktoré som využil ja sám pri celom procese tvorby finálneho užívateľského rozhrania.

Figma

Figma[1] je grafický editor podporujúci vektorovú grafiku. Funguje od roku 2016. Jedná sa o bezplatný nástroj (do istých obmedzení), ktorý je veľmi intuitívny a veľmi rýchlo si tak získal istú časť používateľov konkurenčných grafických editorov zameraných na tvorbu užívateľského rozhrania, akými sú napríklad Sketch[2], AdobeXd[3] či InVision[4]. K dispozícii je aj mobilná aplikácia Figma Mirror v ktorej si viete premietnuť rozpracovaný návrh priamo na vašom mobilnom zariadení. Navyše je dostupná funkcionalita *Prototype* (viz obrázok 4.1), ktorá z obyčajného návrhu vytvorí klikateľný prototyp. Ten môže poslúžiť pri užívateľskom testovaní rozhrania.



Obrázek 4.1: Figma Prototype functionality

Adobe Color

Služba Adobe Colors[5] je online nástroj vhodný pri návrhu farebnej palety a porovnaní rôznych farebných kombinácií. Veľmi užítočnou funkcionalitou je *Color Harmony Rule*[6], ktorý automaticky vytvára kombináciu farieb podľa hlavnej zvolenej farby.

[1] Figma https://www.figma.com
[2] Sketch https://www.sketch.com
[3] Adobe Xd https://www.adobe.com/products/xd.html
[4] InVision https://www.invisionapp.com
[5] Adobe Colors https://color.adobe.com/create/color-wheel
[6] Color Harmonies https://www.tigercolor.com/color-lab/color-theory/color-theory-intro.htm

Dribbble

Dribbble[7] je sociálna sieť pre dizajnérov a kreatívcov, ktorí chcú verejne publikovať svoje výtvory a budovať svoje dizajnové portfolio. Na platforme je nespočetné množstvo návrhov užívateľských rozhraní odpovedajúcich aktuálnym grafickým trendom. Dribbble je vhodným nástrojom pre prvotnú inšpiráciu pri tvorbe užívateľského rozhrania.

Mobbin.design

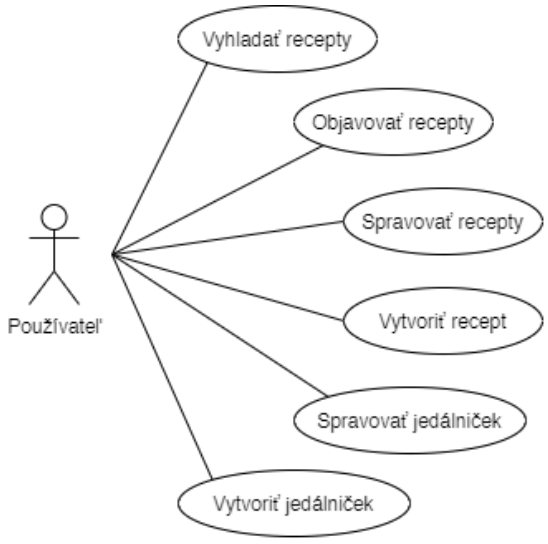
Na rozdiel od Dribbble[8], Mobbin.design predstavuje už skutočne implementované užívateľské rozhrania v rámci populárnych aplikácií, ktoré už v praxi fungujú. Pri každej aplikácií sú dostupné snímky obrazovky a screen-flow.

UI Garage

Stránka UI Garage[9] obsahuje kolekcie najrôznejších návrhových vzorov s príkladmi reálnych aplikácií. Medzi návrhové vzory patrí napríklad *Launch screen*, *Dashboard*, *Ask permissions*, *Settings* a iné, Mobbin.design spolu s UI Garage sú vhodnými nástrojmi pri riešení špecifických problémov v rámci návrhu užívateľského rozhrania. Oba nástroje slúžia pre inšpiráciu a dohľadanie fungujúcich riešení, ktoré už existujú v iných populárnych aplikáciach.

4.3 Diagram prípadov užitia

Pred samotným návrhom aplikácie prebehla analýza aplikácie a identifikácia problémov, ktoré by aplikácia z pohľadu užívateľa mala riešiť. V tomto kroku som vychádzal už zo spomínaného kontextu 1 a zostrojil diagram prípadov užitia, ktorý je možné vidieť na obrázku 4.2. Diagram sa zameriava na najkľúčovejšie prípady užitia a jasne tak pomáha pri návrhu aplikácie zamerať sa na tie najdôležitejšie funkcionality.

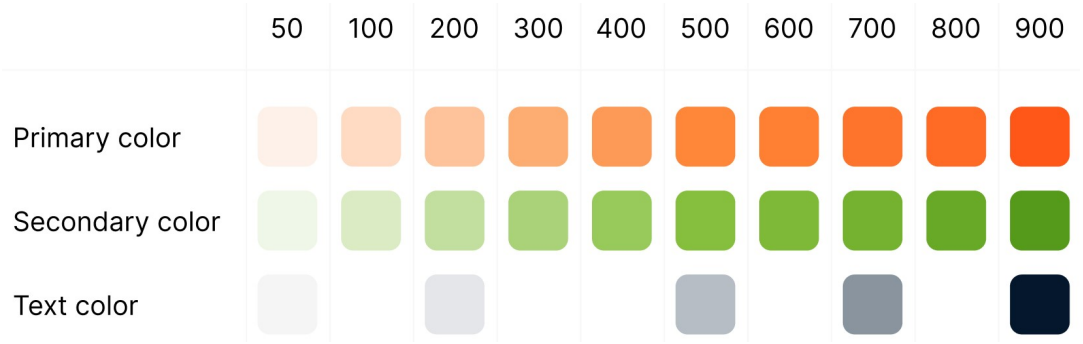


Obrázek 4.2: Diagram prípadov užitia

[7]Dribbble https://dribbble.com
[8]Mobbin.design https://mobbin.design
[9]UI Garage https://uigarage.net

4.4 Voľba farebnej palety

Material Design prichádza s vlastným odporúčaným systémom farieb[10], ktorý definuje výslednú farebnú paletu. Najdôležitejšie je zvoliť si *primary color* a *secondary color*, ktoré majú reprezentovať značku a samotnú aplikáciu. Ďalej *background color* ako farbu pozadia, ale býva dobrým zvykom zvoliť čisto bielu farbu, na ktorej väčšina farebných kombinácií vynikne. V neposlednom rade je dôležitá aj *text color*, ktorá bude dostatočne kontrastovať s predchádzajúcimi farbami tak, aby bol text dobre čitateľný. K vybraným farbám sa následne pomocou online nástrojov[11] vygenerujú jej svetlejšie a tmavšie varianty. Material Design používa pre odtiene stupnicu od 50 do 900 – od najsvetlejšej po najtmavšiu variantu. Nakoniec som sa rozhodol pre špecifickú oranžovú a zelenú farbu, doplnenú o čisto bielu farbu pozadia a šedú respektíve tmavo modrú farbu pre ikony a texty 4.3.



Obrázek 4.3: Zvolená farebná paleta

[10] Material Design - Color System https://material.io/design/color/the-color-system.html
[11] Material Color Tool https://material.io/resources/color/

4.5 Jednotlivé fázy návrhu

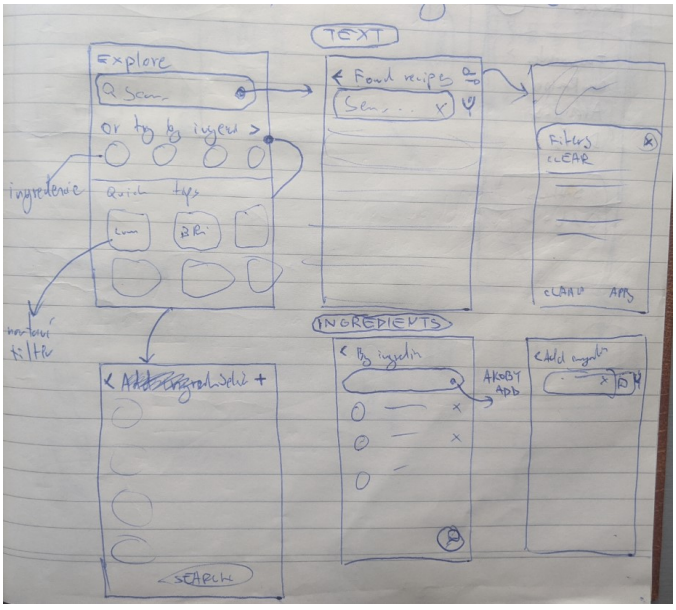
Návrh užívateľského rozhrania (*User Interface* – ďalej ako *UI*) prebiehal v niekoľkých fázach, počas ktorých sa formovalo výsledné rozhranie. Každá fáza istým spôsobom definovala nové požiadavky na aplikáciu, napríklad nutnosť prihlasovania sa do aplikácie, aby bolo možné ukladať užívateľské dáta. Postupným vývojom UI sa taktiež isté funkcionality prehodnotili, zjednodušili a prípadne úplne odstránili za účelom jednoduchého a intuitívneho rozhrania.

Skice

Absolútne prvé návrhy boli nakreslené iba voľne rukou na papier. Návrhy však nevytvárali dokopy jeden celok. Výstupom tejto fázy boli iba **prototypy jednotlivých procesov** demonštrujúce hlavné funkcionality aplikácie – napríklad vyhľadávanie receptov 4.4. Preto nebolo potrebné navrhnúť všetky obrazovky, nakoľko ich funkcionalita je rovnaká – napríklad zoznam užívateľových diét a zoznam užívateľových intolerancií. Oboje sú v hrubom návrhu rovnaké, iba textácia je rozdielna.

Hlavné funkcionality, výzvy a požiadavky, ktoré táto fáza definovala:

- Užívateľský profil - je potrebné, aby sa používateľ do aplikácie prihlásil
- Vyhľadávanie podľa ingrediencií - možnosť odfotiť ingredienciu pre automatické rozpoznanie
- Hlasový vstup - používateľ môže textový vstup zadať vďaka rozpoznávaniu hlasu
- Tvorba vlastných receptov - používateľ si môže vytvoriť vlastné recepty
- Tvorba jedálničku - používateľ si naplánuje jedálniček budúcich dní



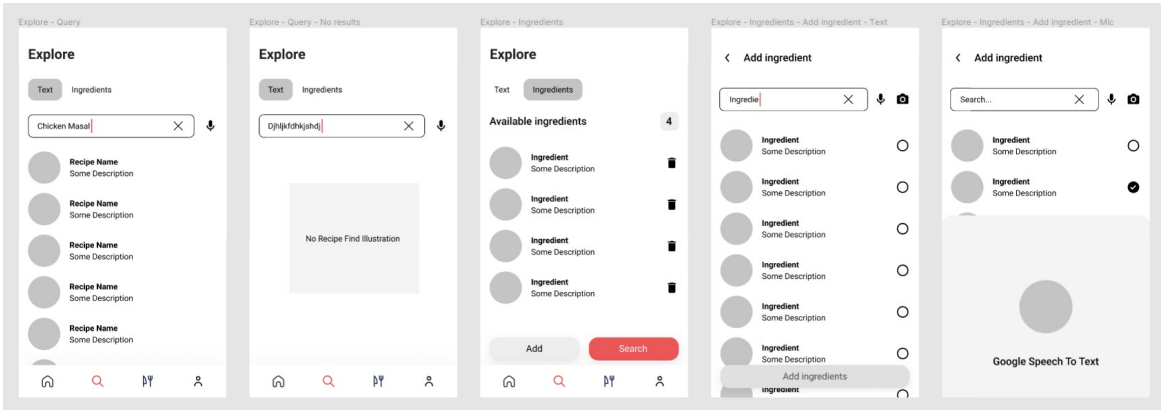
Obrázek 4.4: Prvotný návrh vyhľadávania receptov

Drôtené rámy s nízkou presnosťou

Ďalšia fáza nárvhu UI spočívala v prevedení pôvodných návrhov z papiera do elektronickej podoby vytvorením drôtených rámov s nízkou presnosťou (*low fidelity wireframes*). Každá obrazovka je reprezentovaná jedným wireframom. V tomto štádiu je už aplikácia navrhnutá ako celok a obsahuje všetky obrazovky. Dôležitou súčasťou je aj analýza možných stavov, ktoré môžu nastať v rámci jednej obrazovky – napr. prázdny výsledok pri vyhľadávaní receptov, ak vstupnému reťazcu nevyhovuje žiaden recept. V tomto prípade je konkrétna obrazovka reprezentovaná viacerými wireframes, ktoré definujú správanie obrazovky v rôznych stavoch a prechodmi medzi jednotlivými stavmi – napr. tlačidlo pre pridanie ingredencie zmení stav z *disabled* na *enabled* ak je vybraná aspoň 1 položka. Drôtené rámy s nízkou presnosťou boli vytvorené v programe Figma[12]. Progres spomenutého procesu vyhľadávania receptov je možný vidieť na obrázku 4.5.

Na obázku je taktiež vidieť *bottom navigation menu* tvorené 4 ikonami. Umožňuje používateľovi prechod medzi 4 základnými destináciami aplikácie:

- Domov
- Vyhľadávanie
- Jedálniček
- Užívateľský profil



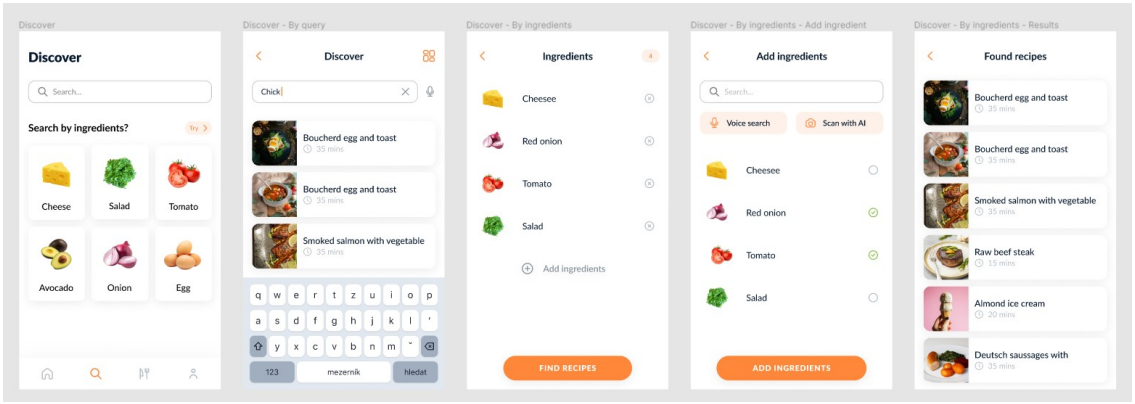
Obrázek 4.5: Upravený návrh vyhľadávania receptov

[12]Odkaz na drôtené rámy s nízkou presnosťou vo Figme https://www.figma.com/file/Okx189Nyewtd7ZGkjEHl7k/?node-id=11%3A124

Drôtené rámy s vysokou presnosťou

Na záver bolo potrebné do návrhov vniesť reálne obrázky, aplikovať textové štýly, fonty, konkrétne textácie a vybranú farebnú paletu z podkapitoly 4.4. Výstupom poslednej fázy sú drôtené rámy s vysokou presnosťou (*high fidelity wireframes*), podľa ktorých sa bude dizajn implementovať do aplikácie. Výsledný produkt by mal byť čo najviac podobný týmto finálnym wireframom. Všetky texty v aplikácií využívajú *font family* **Lato**, spadá pod knižnicu Google Fonts[13], ktorá ponúka bezplatné licencie počítačových písiem.

Návrh vyhľadávania receptov z predchádzajúcej fázy som podrobil užívateľskému testovaniu, ktorého sa účastnili traja užívatelia. U všetkých sa objavili pochybnosti a nejasnosti akým spôsobom vyhľadávať recepty podľa ingredencií. Na základe tejto spätnej väzby som proces vyhľadávania receptov upravil a opätovne ho podrobil užívateľskému testovaniu. Tentokrát sa už zdal byť testerom proces intuitívnejším. Časť finálneho návrhu procesu vyhľadávania receptov je na obrázku 4.6. Vytvorenie drôtených rámov s vysokou presnosťou prebiehalo opäť v programe Figma[14].



Obrázek 4.6: Časť finálneho návrhu vyhľadávania receptov

[13] Google Fonts https://fonts.google.com

[14] Odkaz na drôtené rámy s vysokou presnosťou vo Figme https://www.figma.com/file/Okx189Nyewtd7ZGkjEHl7k/?node-id=206%3A0

Kapitola 5

Návrh systému a výber technológií

V tejto kapitole sa budem venovať zvoleným riešeniam a vybraným technológiam, ktoré v rámci tejto práce pomáhajú napĺňať špecifikované požiadavky a procesy aplikácie.

5.1 Špecifikácia procesov v aplikácií

Pred tým ako sa dostaneme ku konkrétnym riešeniam by som rád zhrnul doterajšie prvotné požiadavky a procesy z hľadiska UI, ktoré vyžadujú priezkum technológií, porovnanie možných riešení a finálny výber vhodného produktu či služby.

Už podľa samotného názvu tejto práce je jasné, že aplikácia má umožniť vyhľadávanie a správu receptov. Existuje nespočetné množstvo receptov a každým dňom vznikajú nové nápady a kombinácie ingrediencií. Vytvorenie vlastnej databázy s receptami by bolo objavovanie kolesa, preto je potrebné spraviť priezkum v rámci existujúcich riešení a služieb, ktoré obsahujú množstvo existujúcich receptov a majú taktiež nad svojou databázom zavedenú architektúru REST API[1]. Je potrebné aby vyhľadávanie receptov bolo možné aj na základe dostupných ingrediencií.

Vyhľadávanie receptov skrz konkrétne ingrediencie navádza na vstup samotných ingredencií, ktoré musí užívateľ nejakým spôsobom do aplikácie zadať. Okrem zadávania ingrediencií pomocou textového reťazca, či hlasovým vstupom, sa ponúka aj využitie rozpoznávania objektov z obrazu. V tomto prípade sa jedná o práve zostrojenú fotku, či vybranú fotografiu z galérie telefónu. Príkladným *use caseom* je, ak užívateľ vytiahne viaceré ingrediencie z chladničky, odfotí ich a aplikácia mu ponúkne recepty, ktoré je možné z dostupných ingredencií pripraviť.

Ovládanie hlasom je stále viac a viac populárne a v dnešnej dobe sa jedná už o akýsi bežný štandard. Zadávanie vstupného reťazca hlasovým vstupom je ďalším požiadavkom, ktorým by táto aplikácia mala disponovať.

Užívateľ si môže recepty ukladať do obľúbených a vytvárať svoje vlastné recepty. Taktiež si môže v rámci aplikácie zaznamenať svoje intolerancie a diéty. Aby boli dáta dostupné aj po prípadnom zmazaní dát, či na inom zariadení, je nutné aby aplikácia vyžadovala prihlásenie do užívateľských účtov. Ďalej je potrebné údaje ukladať niekam na server, kde budú zálohované. V neposlednom rade je treba dbať na bezpečnosť užívateľových dát a prihlasovacích prístupov.

[1]Representational State Transfer Application Programming Interface (REST API) - aplikačné programové rozhranie, ktoré pracuje s REST architektúrou. REST architektúra prezentuje spôsob, pomocou ktorého je možné čítať, zapisovať, editovať a mazať informácie na serveri pomocou HTTP volaní.

5.2 Rest API

Jedna z najdôležitejších problematík v rámci riešenej aplikácie je zdroj dát. Pod zdrojom dát rozumieme recepty, ingrediencie, nutričné hodnoty a iné. Špecifickým požiadavkom je taktiež vyhľadávanie receptov pomocou vstupných ingredenicí. Pri voľbe REST API služby je dôležité dbať aj na metriky, ktorými jednotlivé služby disponujú. Dôležité sú aj obmedzenia z hľadiska vyťaženosti a počtu vykonaných HTTP požiadavkov v priebehu minúty, dňa či mesiaca. Každá služba to môže mať definované inak. V rámci priezkumu dostupných služieb disponujúcich REST API nad svojou databázou receptov som narazil na niekoľko možností. Za zmienku stoja služby Edamam [2], BigOven [3] a Spoonacular Food API [4]. Napokon som sa rozhodol pre poslednú zmieňovanú službu, **Spoonacular Food API**.

Spoonacular Food API

Spoonacular Food ponúka komplexné potravinové API. V databáze existuje viac než 5000 receptov, zložených z viac než 2600 ingrediencií. Ku každému receptu respektíve ingredencii sú k dispozícií detailne zanalyzované nutričné hodnoty. API umožňuje vyhľadávanie receptov na základe vstupného reťazca či zadaním určitých ingrediencií.

Spoonacular Food API je do istých limitov bezplatná služba. Pri registrácií developer obdrží API kľúč, ktorý je nemenný a prikladá sa ku každému jednému HTTP požiadavku. V bezplatnom programe je možné vykonať maximálne 150 requestov denne, zvyšné požiadavky ostanú neobslúžené. Z hľadiska vývoja je to dostačujúci limit a počas práce na aplikácií som ho nikdy nevyčerpal. Avšak v budúcnosti, z hľadiska zabehnutého produktu, ktorý by mal denne aktívnych niekoľko desiatok užívateľov, by bol tento limit nedostačujúci a vyžadoval by si prechod na platený plán. Platené plány sú však stále cenovo dostupné a náklady by z časti mohli pokryť výnosy z reklamných banerov v aplikácií.

Diéty a intolerancie potravín

Systém zohľadňuje desať rôznych druhov diét a dvanásť rôznych intolerancií potravín.

Diéty:

- **Gluten Free -** bezlepková diéta, človek nekonzumuje pšenicu, jačmeň, raž a iné obilniny a potraviny obsahujúce lepok
- **Ketogenic -** keto diéta je založená na pomere prijatých živín, nešpecifikuje konkrétne potraviny a ingrediencie. Všeobecne sú akceptovateľné jedlá s vysokým obsahom tukov a bielkovín, naopak neakceptovateľné sú jedlá s vysokým obsahom sacharidov
- **Vegetarian -** vegetariánska diéta nesmie obsahovať žiadne mäso alebo vedľajšie mäsové výrovky ako sú kosti, bujóny, želatiny
- **Lacto-Vegetarian -** laktovegetarián je vegetarián, ktorý nekonzumuje mäso, ryby ani vajcia, ale konzumuje mliečne výrobky.

[2] Edamam https://developer.edamam.com/food-database-api-docs
[3] BigOven https://api2.bigoven.com
[4] Spoonacular https://spoonacular.com/food-api

- **Ovo-Vegetarian -** ovo vegetarián je vegetarián, ktorý nekonzumuje mäso, ryby ani mliečne výrobky, ale konzumuje vajcia.
- **Vegan -** vegánska strava nesmie obsahovať mäso, vedľajšie mäsové výrobky a ani žiadne živočíšne produkty ako sú vajcia, mliečne výrobky alebo med.
- **Pescetarian -** pescetariánska diéta spočíva v absencií konzumácie mäsa a mäsových výrobkov, naopak morské plody a ryby sú akceptovateľné
- **Paleo -** paleotická diéta vyraďuje respektíve obmedzuje poľnohospodárske produkty ako obilniny, strukoviny, pečivo a mliečne výrobky. Jedálniček sa skladá hlavne z mäsa, rýb, zeleniny, ovocia, húb, orechov a vajec. Táto diéta je tiež prezývaná ako diéta doby kamennej [5]
- **Primal -** veľmi podobná paleotickej diéte, s výnimkou povolenia mliečnych výrobkov
- **Whole30 -** jedná sa o kratkodobý (30 dní a viac) nutričný reštart organizmu, ktorého cieľom je skoncovať s nezdravými stravovacími návykmi, obnovenie zdravého metabolizmu a tráviaceho traktu. Zakázané sú hlavne pridané sladidlá, nezdravé a tučné jedlá, poľnohospodárske produkty, siričitany či alkohol.

Intolerancie:

- **Dairy -** mliečne výrobky
- **Egg -** vajcia
- **Gluten -** lepok
- **Grain -** obilie
- **Peanut -** arašidy
- **Seafood -** morské plody
- **Sesame -** sezam
- **Shellfish -** mäkkýše
- **Soy -** sója
- **Sulfite -** siričitan
- **Tree Nut -** orechy
- **Wheat -** pšenica

Dôležité endpointy

Služba Spoonacular Food má ku svojmu API vypracovanú podrobnú dokumentáciu[5], ktorá objasňuje jednotlivé jednotlivé endpointy a rozdeluje ich do kategórií. Celá dokumentácia je dostupná aj vo formáte JSON, ktorý je možný importovať do aplikácie Postman[6], ktorú som použil na testovanie a priezkum endpointov. API fungujú na princípe požiadaviek (request) a odpovedí (response). Aplikácia odošle na server požiadavok a server reaguje odpoveďou. API Endpoint (koncový bod aplikačného rozhrania) je špecifická URL adresa, ktorá definuje požiadavok z aplikácie mierený na server za účelom získania odpovedi. Všetky požiadavky musia obsahovať parameter `apiKey`, v ktorom je predávaný developerský kľúč.

[5] Spoonacular Food API dokumentácia https://spoonacular.com/food-api/docs
[6] Postman https://www.postman.com

- `GET https://api.spoonacular.com/recipes/complexSearch`

  Vyhľadávanie receptov podľa vstupného reťazca - koncový bod používaný pre komplexné respektíve filtrované vyhľadávanie receptov. V povinnom parametri `query` je predávaný vstupný reťazec zadaný používateľom. V prípade aktívnych diét či intolerancií sú použité aj parametre `diet` a `intolerances` pre zohľadnenie výsledkov vyhľadávania.

- `GET https://api.spoonacular.com/recipes/findByIngredients`

  Vyhľadávanie receptov podľa ingredencií - endpoint používaný pre vyhľadávanie receptov podľa zadaných vstupných ingredencií. V povinnom parametri `ingredients` sa predáva zlúčený reťazec názvov vstupných ingrediencií, ktoré musia byť oddelené čiarkou. Výsledkom je zoznam receptov, ktoré obsahujú dané ingrediencie.

- `GET https://api.spoonacular.com/recipes/random`

  Zoznam náhodných respektíve populárnych receptov - vo finálnom návrhu užívateľského rozhrania 4.5 sú na domovskej obrazovke karty s náhodne odporúčanými receptami. Tento endpoint je použitý ako zdroj dát pre odporúčané recepty pre používateľa. V nepovinnom parametri `number` (dátový typ integer) sa zadáva počet požadovaných receptov v odpovedi. Aplikácia vyžaduje v odpovedi presne 5 náhodne odporúčaných receptov.

- `GET https://api.spoonacular.com/recipes/{id}/information`

  Detail receptu - koncový bod pre získanie detailu konkrétneho receptu. Recept je určený povinným parametrom `id`, ktorý odpovedá jedinečnému identifikátoru daného receptu. Aplikácia využíva taktiež nepovinný booleanovský parameter `includeNutrition` pre získanie nutričných hodnôt nachádzajúcich sa v recepte.

- `GET https://api.spoonacular.com/recipes/{id}/similar`

  Zoznam podobných receptov - za zmienku stojí aj koncový bod pre vyhľadanie podobných receptov, ktoré by mohlo byť zahrnuté v ďalších aktualizáciach aplikácie. Na obrazovke detailu receptu by mohla byť možnosť zobraziť podobné recepty, na základe ktorej by sa odosielal požiadavok na tento koncový bod. Povinný parameter `id` definuje identifikátor konkrétneho receptu, na základe ktorého budú dohľadané podobné recepty.

5.3 Firebase

Firebase[7] je platforma vyvinutá spoločnosťou Google pre vývoj a správu mobilných a webových aplikácií. Firebase bola pôvodne nezávislá spolosčnosť založená v roku 2011, ale v roku 2014 ju odkúpil gigant Google, ktorý ju neustále rozširuje o nové služby a produkty, ktoré napomáhajú pri vývoji aplikácií. K Firebase projektu sa je možné pripojiť okrem Android a iOS aplikácie aj webovú aplikáciu.

Platforma sa postupom času reformovala a dnes rozdeľuje svoje služby medzi 4 základné skupiny nástrojov:

- **Build -** pomáha pri budovaní aplikácie, obsahuje možnosti pre autentikáciu užívateľov, real-time databázu či úložisko dát
- **Release & Monitor -** zaznamenáva prípadné pády aplikácie a ponúka rôzne možnosti a konfigurácie pri testovaní
- **Analytics -** súhrn spracovaných dát a štatistík, ktoré aplikácia dosahuje
- **Engage -** nástroje pre udržanie publika, akvizície nových používateľov a speňaženie aplikácie v prípade prítomnosti reklám

Autentifikácia

Nástroj pre jednoduché prihlásenie pomocou ľubovoľnej platformy. Cieľom Firebase Authentication je uľahčiť vytváranie zabezpečených autentifikačných systémov a zároveň vylepšiť možnosti registrovania a prohlasovania koncových používateľov. Vďaka poznaniu identity používateľa je možné ďalej pracovať a uchovávať jeho údaje v rámci systému. Podporuje autentifikáciu pomocou hesiel, telefónnych čísiel a taktiež podporu pre prihlásenie tretích strán od populárnych federovaných poskytovateľov identity ako sú napríklad Google, Facebook, Twitter, Github a ďalší. Firebase Authentication využíva štandardy ako OAuth 2.0 a OpenID, takže ju možno ľahko integrovať prípadne aj do vlastného backendového systému.

Real time database

Firebase Realtime Database je NoSQL databáza hostená v cloude, v datacentrách od Google rozložených po celom svete. Dáta sa ukladajú vo formáte JSON a synchronizujú sa v reálnom čase s každým pripojeným klientom. Synchronizácia dát však môže prebehnúť aj jednorázovým vyčítaním dát, ktoré eliminuje neustále pripojenie klientov na databázu. Počet aktívne pripojených klientov je limitovaný maximálne na 100 zariadení súčasne v rámci neplateného plánu.

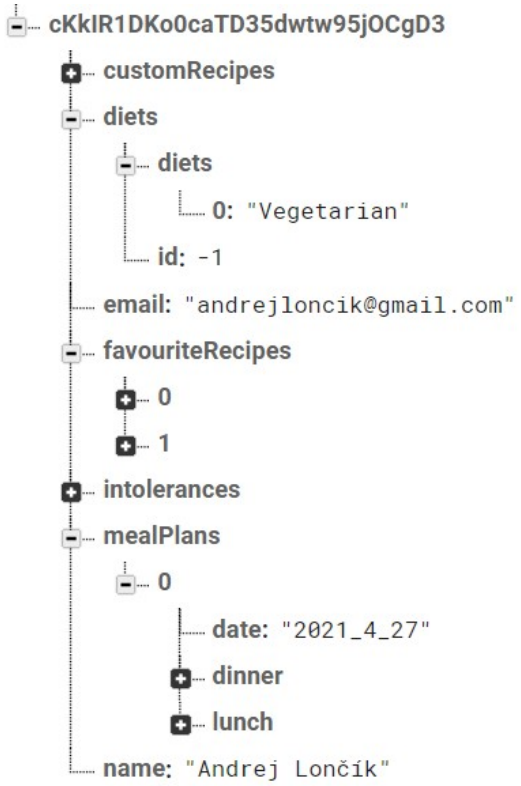
Databáza bude slúžiť predovšetkým pre uchovávanie užívateľských dát. Každý prihlásený používateľ, ktorý prešiel autentizáciou, má v ramci systému od Firebase pridelený svoj unikátny užívateľský identifikátor (User UID), ktorý jeho účet definuje. Pod týmto UID je v poli užívateľov uchovávaný jeho užívateľský obsah z aplikácie. Štruktúra uchovávaných dát v databéze je zobrazená na obrázku 5.1.

Medzi užívateľský obsah patrí:

- **Meno -** reťazec
- **Email -** reťazec

[7] Firebase https://firebase.google.com/

- **Diéty -** pole reťazcov, kde každý reťazec odpovedá konkrétnej užívateľskej diéte
- **Intolerancie -** pole reťazcov, kde každý reťazec odpovedá konkrétnej užívateľskej intolerancií
- **Obľúbené recepty -** pole objektov triedy Recept, prebraného zo Spoonacular Food API
- **Vlastné recepty -** pole objektov triedy Recept, vytvoreného v aplikácií
- **Jedálniček -** pole objektov triedy MealPlan, ktorý predstavuje jeden kalendárny deň v rámci jedálničku. Parametre `breakfast`, `lunch` a `dinner` obsahujú pole receptov



Obrázek 5.1: Štruktúra uchovávaných užívateľských dát v databáze

Crashlytics

Firebase Crashlytics je nástroj na hlásenie chýb a zlyhaní aplikácií v reálnom čase. Zachytáva každý pád aplikácie a prikladá záznam logu, z ktorého je možné zistiť príčinu pádu aplikácie. Taktiež obsahuje aj informácie o zariadení, verzi operačného systému a verzii samotnej aplikácie. Ak sa rovnaké zlyhanie aplikácie prejavilo u viacerých užívateľov na rôznych zariadeniach, Firebase urguje developera upozorneniami, kde označí zlyhanie ako *Trending Issue*. Developer by mal tak čo najskôr vydať novú verziu s opravami, aby sa predišlo ďalším pádom aplikácie.

Analytics

Srdcom nástroja Firebase sú Google Analytics, bezplatné riešenie pre meranie a získavanie prehľadov o používaní aplikácie a angažovaní nových užívateľov. Samotné funkcie a metriky Google Analytics sú veľmi rozsiahlou tématikou, presahujú do marketingu a následného optimalizovania vytvoreného produktu.

5.4 Počítačové videnie

Podpora počítačového videnia a rozpoznávania objektov z obrazu má v aplikácií využiteľnosť pri zadávaní ingrediencií. Príklad užitia, používateľ si chce uvariť nejaký recept, ale má obmedzené množstvo surovín. Vytiahne dostupné ingrediencie z chladničky, zo špajze a položí ich na kuchynskú linku. Vezme do rúk mobil a v ideálnom prípade aplikácia rozozná z jedinej fotky všetky dostupné ingrediencie. Následne vyhľadá vyhovujúce recepty. Existuje niekoľko služieb, ktoré ponúkajú rôzne metódy počítačového videnia a výsledné formy rozpoznávania z obrazu (viz. obrázok 5.2). Medzi najrozšírenejšie patria:

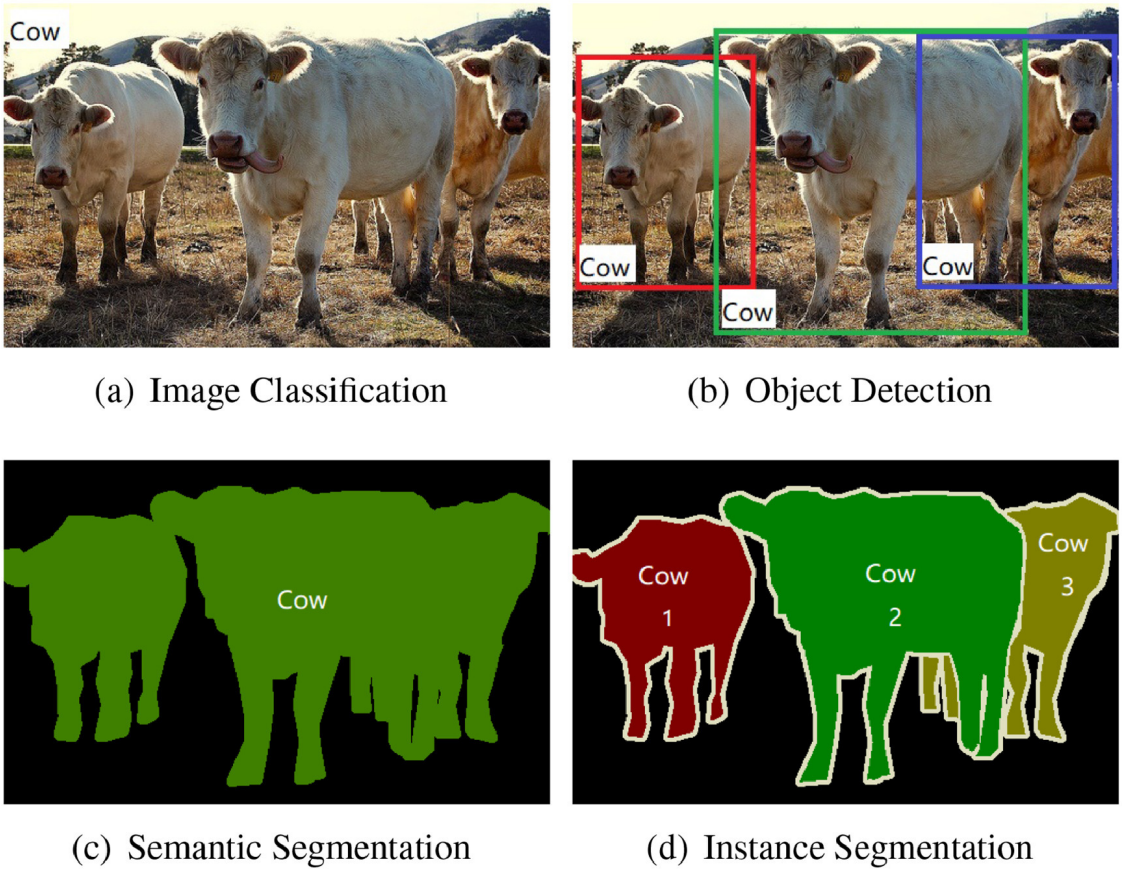
- **Text recognition -** rozpoznanie textu
- **Face detection -** rozpoznanie tvárí a prípadné detekovanie emócií
- **Image classification -** nazývané aj image labeling, opisuje čo sa nachádza v rozpoznávanom obraze, hlavným výstupom je pole reťazcov
- **Object detection -** detekovanie a lokalizácia rozpoznaných objektov, každý výstupný reťazec obsahuje aj `bounding boxes` súradnice pre lokalizáciu obdĺžnika, v ktorom sa nachádza konkrétny objekt
- **Object tracking -** object detection v pohybe, to znamená v rámci videa
- **Semantic segmentation -** priraďuje každému pixelu farbu podľa názvu rozpoznanej kategórie objektu
- **Instance segmentation -** priraďuje každému pixelu farbu podľa inštancie rozpoznaného objektu

Na základe pôvodeného požiadavku, rozpoznávanie ingrediencií z obrazu, v tejto práci postačí využiť metódu **Image classification**. V nasledujúcih podkapitolách sa zameriam na jednotlivé služby, ktoré touto funkcionalitou disponujú a je možné ich v rámci možností implementovať do Android aplikácie.

Amazon Rekognition

Amazon Rekognition[8] je služba od Amazonu spadajúca pod balík **AWS** – Amazon Web Services. Ponúka rôzne metódy rozpoznávania obrazu a je zadarmo do istého limitu (5000 obrázkov ročne). Aplikáciu je nutné registrovať v rámci AWS a je dostupná aj základná dokumentácia. Nevýhodou je však nízka popularita v rámci implementácie týchto služieb na platformu Android s čím súvisí aj absencia článkov a exemplárnych príkladov tretích strán. Služba je viac mierená na webový a desktopový vývoj.

[8] Amazon Rekognition https://docs.aws.amazon.com/rekognition/latest/dg/labels.html



(a) Image Classification (b) Object Detection

(c) Semantic Segmentation (d) Instance Segmentation

Obrázek 5.2: Porovnanie vybraných metód počítačového videnia, prebrané z [12]

Tensor Flow

TensorFlow[9] je machine learning platforma od firmy Google. Ponúka možnosť vytvoriť a natrénovať si vlastný model pre rozpoznávanie ingrediencií. Vzhľadom na širokú množinu existujúcich ingrediencií sme po konzultácií s vedúcim usúdili, že ísť cestou tvorby vlastného modelu rozhodne nie je najlepší nápad.

Fritz AI

Fritz AI[10] je pomerne neznáma súkromná firma venujúca sa oblasti Machine Learning. Dokumentácia je rozsiahla a ponúka niekoľko príkladov mobilných aplikácií komunikujúcich s ich rozhraním. Image Labeling navyše prebieha v real-time čase a je dohľadateľných aj niekoľko článkov tretích strán. Nevýhodou je však cena tejto služby, ktorá začína od 179$ mesačne.

[9] TensorFlow https://www.tensorflow.org/lite/examples/image_classification/overview
[10] Fritz AI https://docs.fritz.ai/develop/vision/image-labeling

Firebase ML Kit

Firebase Machine Learning Kit[11] je nástroj od Firebase, ktorý ako som už spomínal, taktiež spadá pod firmu Google. Obsahuje podrobnú dokumentáciu a taktiež je dohľadateľných množstvo článkov a publikácií, ktoré demonštrujú funkčnosť tejto služby. K dispozícií sú dve možnosti image labelingu:

- **On-device -** prebieha offline na zariadení, je zadarmo, ale obsahuje približne 400+ štítkov
- **Cloud -** obrázok sa odosiela na server, ktorýho spracuje. Je teda potrebné byť online, prvých 1000 použití v mesiaci je zadarmo a obsahuje viac než 10000+ štítkov

Napokon som sa rozhodol práve pre využitie nástroja **Firebase Machine Learning Kit** a to najmä z hľadiska dostupnosti, dokumentácie a ceny. Využijem variantu Cloud, pri ktorej Firebase ML Kit disponuje už natrénovaným modelom a rozoznáva viac než 10000+ štítkov. Vzhľadom na širokú škálu možných ingrediencií, bude lepšie pre prácu využiť už existujúci model, ako vytvárať a trénovať svoj vlastný pre klasifikáciu ingrediencií.

[11] Firebase Machine Learning Kit https://firebase.google.com/docs/ml-kit/label-images

Kapitola 6

Implementácia

Po návrhu užívateľského rozhrania, analýze systému a výberu technológií prichádza na radu samotná implementácia mobilnej aplikácie, ktorú priblížim na nasledujúcich stranách. V jednotlivých podkapitolách sú opísané prístupy, problémy a riešenia, s ktorými som sa počas implementácie potýkal.

Aplikácia bola implementovaná vo vývojovom prostredí Android Studio 3.3, prevažne v programovacom jazyku Kotlin 3.3. Pre verzovanie kódu som použil systém Git a službu GitHub[1].

6.1 Architektúra MVVM

Aplikáciu som vyvíjal na základe architektúry MVVM, ktorá je priamo odporúčaná od Google pre vývoj[2]. Názov MVVM je odvodený z úplneho názvu základných prvkov architektúry, **Model**–**View**–**ViewModel**. Najdôležitejšou myšlienkou architektúri je rozdelenie zodpovedností a oddelení UI od biznis logiky. Mnohí, hlavne začínajúci, programátori sa dopúšťajú tej chyby, že všetok kód je napísaný v zdrojových kódoch fragmentov či aktivít. Fragmenty v tom prípade okrem spracovania vstupných interakcií musia riešiť aj sieťové volania, ukladanie do databáz či prezentačnú logiku. Výsledkom je neprehľadný kód, duplikovanie kódu, preťažené UI vlákno a rada problémov vyplývajúcich zo životného cyklu fragmentu respektíve aktivít 3.2.

Architektúra sa skladá z troch prvkov, jej princíp je znázorňený na obrázku 6.1:

- **Model -** mal by obsahovať všetkú logiku aplikácie. Spracovanie a ukladanie dát, prepojenie s databázou, prístup k internému úložisku, komunikáciu so serverom či prípadné zložitejšie výpočty. Model je založený na princípe návrhového vzoru *Singleton*, takže v rámci aplikácie existuje len jediná inštancia konkrétneho modelu. Model pracuje s ViewModelom za účelom ukladania a získania dát.

- **View -** predstavuje užívateľské rozhranie, jednu konkrétnu obrazovku. Jeho úlohou je spracovávať vstupnú interakciu od užívateľa a preposielať ju ViewModelu, spozorovať následnú zmenu dát a aktualizovať vizuál. View by mal byť čo najviac odľahčený a neobsahovať žiadnu komplikovanú výpočtovú logiku.

- **ViewModel -** slúži ako prostredník medzi View a Modelom. Príjma a reaguje na vstupné akcie volané z View, spracuje ich a prípadne preposiela zmenu hodnôt do

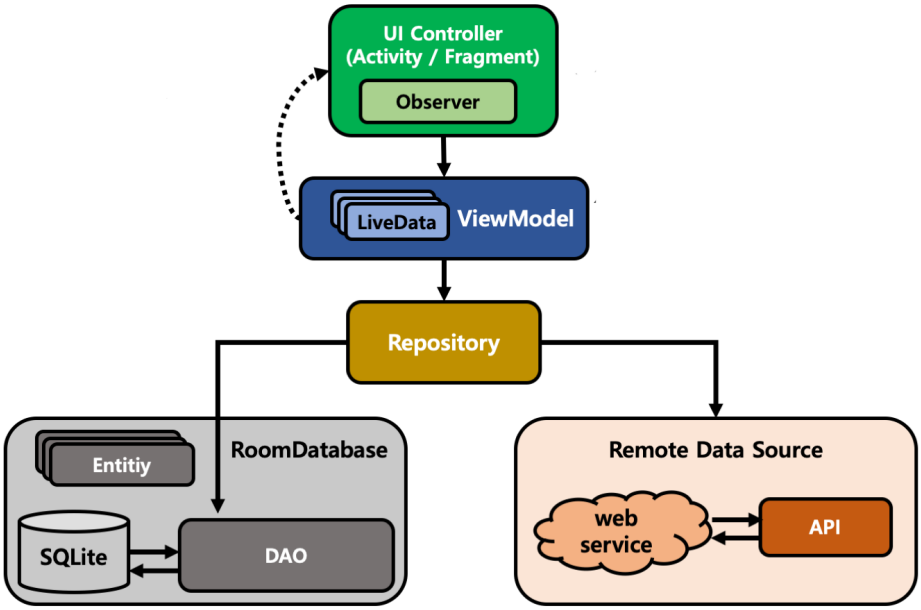
[1] GitHub repozitár https://github.com/Lonchi78/Bakalarka
[2] Android app architecture MVVM https://developer.android.com/jetpack/guide

Modelu. Z Modelu naopak prúdia aktualizované dáta, nad ktorými prípadne aplikuje prezentačnú logiku a prepošle ich naspäť do View. O prúdenie dát sa stará trieda LiveData[3] (prípadne MutableLiveData), ktoré automaticky upozorňujú *observer* na zmeny.

Príkladom je **vyhľadávanie receptov podľa vstupného reťazca** zadaného užívateľom. View `DiscoverByQueryFragment` spracováva zmenu reťazca z komponenty EditText. Ak dĺžka textu presahuje tri znaky, volá funkciu z ViewModela `DiscoverByQueryViewModel`. ViewModel ďalej volá funkciu z Modelu `RecipesRepository`, ktorá na pozadí vykoná požiadavok a získa zo serveru zoznam receptov. Napokon pomocou LiveData výsledok prúdi cez ViewModel naspäť do View, ktorý výsledok vyhľadávania zobrazí používateľovi.

Osobne sa držím pravidla, kde každý existujúci View (teda fragment alebo aktivita) má svoj jeden vlastný ViewModel a ku každej entite, ktorá v systéme vystupuje, prináleží jeden Repository, teda Model. ViewModel môže komunikovať s viacerými Modelmi. Modely môžu medzi sebou taktiež komunikovať.

S architektúrou aplikácie úzko súvisí aj *dependency injection*, na ktorú používam knižnicu **Dagger**[4]. V preklade sa jedná o injekciu závislostí, čo je dosť výstižný popis. Dagger sa používa k implementácií vzájomných závislostí medzi triedami respektíve časťami MVVM, s ktorými pracuje ako s modulami. Základným článkom pri nastavení Daggeru je AppComponent, kde sú definované všetky moduly. Vo vyššie uvedenom príklade figurujú dve časti Daggeru, modul `DiscoverByQueryModule`, ktorý spája View s ViewModelom a definícia metódy `provideRecipesRepository()`, ktorá poskytuje inštanciu *singletonu*.



Obrázek 6.1: Architektúra MVVM v Androide, prebrané z [10]

[3]Live Data https://developer.android.com/topic/libraries/architecture/livedata
[4]Dagger dependency injector https://github.com/google/dagger

6.2 Aktivity a Fragmenty

O aktivitách a fragmentoch sme sa bavili už v kapitole 3.2. Od zavedenia *Navigation Component*[5] Google odporúča dodržovať *Single acitivity prístup* [7], kde je v rámci celej aplikácia iba jediná aktivita a zvyšok je realizovaný cez fragmenty. Ja osobne sa snažím dodržovať čo najnižší možný počet aktivít, avšak uprednostňujem logické rozdelenie aplikácie do viacerých aktivít podľa okolností a účelu.

Aplikácia obsahuje celkom **7 aktivít** a **36 fragmentov**, z nich väčšina spadá práve pod jednu z aktivít. Každá aktivita resp. fragment má svoj ViewModel. `SplashActivity` je prostá aktivita zobrazujúca sa pri spúštaní aplikácie. Následne sa užívateľ dostane do `LoginActivity`, v ktorej sa musí prihlásiť. Spolu s `AboutActivity` to sú jediné obrazovky, do ktorých sa dostane neprihlásený používateľ. Aktivita About obsahuje stručné informácie o aplikácií a autorovi. Po prihlásení sa pri prvom spustení otvorí `OnboardingActivity`, tá slúži ako stručný návod k aplikácií. Detekovanie prvého spustenia je uložené v Shared Preferences pod kľúčom `first.start` s východzou hodnotou `true`. Akonáhle sa spustí aktivita Onboarding, hodnota sa prepíše na `false` a pri ďalšom spustení sa presmerovanie na Onboarding aktivitu odignoruje na základe nevyhovujúcej podmienky.

`MainActivity` je najkomplexnejšou aktivitou v rámci celej aplikácie, spravuje bottom navigation menu a 14 fragmentov. Vyžaduje prihláseného používateľa a v prípade odhlásenia alebo neautorizovanej akcie stačí zničiť túto aktivitu a prejsť do LoginActivity. Netreba sa tak starať o správne ukončenie všetkých fragmentov v zásobníku, ako v prípade *single activity* prístupu.

Proces tvorby vlastného receptu sa skladá z 12 obrazoviek (fragmentov), čo je dostatočne veľké množstvo aby sa jednalo o samostatnú aktivitu `CreateRecipeActivity` s oddeleným navigačným grafom pre možné destinácie (fragmenty).

Posledná – `DiscoverByIngredientsActivity` – ako už názov predpovedá, má na starosti proces vyhľadávania receptov podľa ingrediencií. Súčasťou tohto procesu je spustenie fotoaparátu v rámci aplikácie. To bol hlavný dôvod, prečo som sa rozhodol túto aktivity separovať od MainActivity.

Pred implementáciou konkrétnych fragmentov a aktivít som vytvoril **abstraktné triedy** `BaseFragment`, `BaseActivity` a `BaseViewModel`, ktoré sa starajú o správne vytvorenie view modelu a nastavenie layoutov. Okrem toho obsahujú zopár užítočných funkcií ako napríklad `showProgressDialog()` a `showErrorSnackbar()`.

6.3 Lokálna databáza a uchovávanie dát

Jedným z hlavných požiadavkov aplikácie je schopnosť fungovať bez stálého internetového pripojenia. V offline režime síce užívateľ nemôže vyhľadávať nové recepty, ale má aspoň prístup ku svojím obľúbeným a vytvoreným receptom. Tieto a ostatné údaje z podkapitoly 5.3 sú ukladané v lokálnej databáze aplikácie. Android disponuje SQLite databázou, ktorá je uchovaná v internom úložisku.

Pri práci s SQLite databázou som použil knižnicu Room[6], ktorá je súčasťou balíka Android Jetpack. Room poskytuje abstraktnú vrstvu nad SQLite, ktorá umožňuje plynulý prístup k databáze bez straty na výkone. Pri kompilácií overuje platnosť SQL dotazov, čo zabraňuje prípadným neskorším pádom aplikácie. Taktiež je kompatibilná s triedou Live-

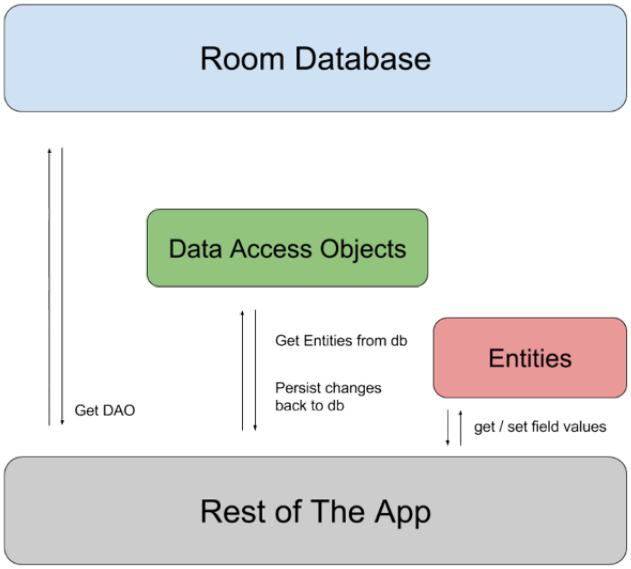
[5] Android Navitgation Component https://developer.android.com/guide/navigation/navigation-getting-started

[6] Room https://developer.android.com/training/data-storage/room

Data, ktoré pri zmene pozorovaných dát automaticky doručí aktualizované hodnoty. Room architektúra (obrázok 6.2) sa skladá z troch vrstiev:

- **Database -** samotná databáza tvoriaca prístupový bod k perzistentným dátam
- **DAO (Data Access Objects) -** rozhranie s metódami definujúce SQL príkazy
- **Entities -** rezprezentuje tabuľky v databáze

S využitím Room databázy aplikácia získa prístup k DAO rozhraniu. Aplikácia pomocou DAO pristupuje k entitám a ich modifikáciam. Následne z konkrétnej entity je aplikácií umožnené čítať a zapisovať hodnoty.



Obrázek 6.2: Štruktúra Room databáze, prebrané z [4]

V ukážke kódu 6.1 je DAO rozhranie pre entitu `MealPlan`. Pomocou tohto rozhrania aplikácia pristupuje k entite `MealPlan`, z ktorej následne číta a zapisuje hodnoty. Anotácia `@Dao` knižnici naznačuje, že sa jedná o DAO. Pred implementáciou všetkých DAO rozhraní som si rovnako ako pri fragmentoch a aktivitách vytvoril rodičovské rozhranie `BaseDao`, ktoré obsahuje najzákladnejšie metódy ako `INSERT`, `UPDATE`, `DELETE` a `REPLACE`. Anotácia `@Query(...)` signalizuje, že sa jedná o SQL dotaz, ktorý je následne definovaný v zátvorkách. Metódy môžu obsahovať argumenty, napríklad `getMealPlan(date: String)`, ktoré je možné aplikovať v SQL dotaze za špeciálnym znakom `:`. Anotácia `@Transaction` predstavuje transakciu. Návratové hodnoty metód môžu byť obalené v triedach Single alebo LiveData pre asynchrónne prevedenie.

```
@Dao
interface MealPlanDao : BaseDao<MealPlan> {

    @Query("SELECT * FROM MealPlan")
    fun getAllMealPlans(): LiveData<List<MealPlan>>
```

```
    @Query("SELECT * FROM MealPlan")
    fun getAllMealPlansBlocking(): List<MealPlan>

    @Query("SELECT * FROM MealPlan WHERE date = :date")
    fun getMealPlan(date: String): Single<List<MealPlan>>

    @Query("SELECT * FROM MealPlan WHERE date = :date")
    fun getMealPlanBlocking(date: String): List<MealPlan>

    @Query("DELETE FROM MealPlan")
    fun deleteAllMealPlans()

    @Transaction
    fun saveMealPlan(mealPlan: MealPlan) {
        insert(mealPlan)
    }

    @Transaction
    fun saveAllMealPlans(mealPlans: List<MealPlan>) {
        insertAll(mealPlans)
    }
}
```

Výpis 6.1: Ukážka MealPlanDao rozhrania, ktoré sa využíva pri práci s jedálničkom

Entity

V aplikácií sa vyskytuje celkom 7 entít, ktoré sú uchovávané v SQLite databáze. Pokiaľ je entita `User` neprázdna tak obsahuje informácie o prihlásenom používateľovi. V opačnom prípade sa jedná o neprihláseného používateľa. Entita `Diets` obsahuje pole reťazcov, ktoré odpovedajú používateľovým diétám. Obdobne `Intolerances` napĺňajú rovnakú funkciu pre zoznam intolerancií. Entity `RecipeFavourite` a `RecipeCustom` vyplývajú z triedy `Recipe` a sú si do istej miery veľmi podobné. Prvá z dvojice prezentuje obľúbený recept z Rest API. Druhý entita odpovedá vlastnému receptu používateľa, ktorého štruktúra sa v prípade ďalších aktualizácií môže rozšíriť o nové parametre. Ďalej entita `MealPlan` z vyšie uvedeného príkladu, predstavuje jeden stravovací deň. Poslednou entitou je `Filter`, ktorá ukladá nastavenie filtrovacích parametrov pri vyhľadávaní receptov. Štruktúru jednotlivých entít nájdete v prílohe B.1.

6.4 Komunikácia so serverom

Aplikácia pracuje s dvoma rôznymi vzdialenými úložiskami. Prvým je služba Spoonacular food API 5.2, ktorá slúži ako zdroj dát pre vyhľadávanie receptov. Druhým je Firebase Realtime Database 5.3, ktorá plní funkciu zálohovania užívateľských údajov. S oboma sa však pracuje rozdielnym spôsobom.

Komunikácia s REST API

Komunikácia so serverom pomocou REST API 5.2 je založená na protokole HTTP a používa komunikačné endpointy. Pre vykonanie komunikácie je potrebné odoslať požiadavok na konkrétnu adresu a spracovať odpoveď. Pre tieto účely som využil knižnicu Retrofit[7], ktorá zjednodušuje komunikáciu so serverom. Dáta z odpovede na požiadavok sú vo formáte JSON a je potrebné ich transformovať na objekty. Použil som knižnicu Moshi[8], ktorá je doporučená a kompatibilná s Retrofitom.

V ukážke kódu 6.2 je metóda `getRecipeDetail()` z rozhrania **RestApi.kt**, ktorá získava detail receptu na základe identifikátoru `id`. Anotácia `@GET` definuje HTTP metódu a za ňou nasleduje v zátvorke adresa endpointu. Je to len prefix, ktorý knižnica pridá ku základnej Base URL (základná URL). Tá sa zadáva už pri nastavovaní samotného Retrofitu. V ukážke môžeme vidieť, že parameter id má pred sebou anotáciu `@Path`. Táto anotácia knižnici vraví, že má tento parameter zameniť na miesto zátvoriek vo vyššie spomínanej adrese endpointu. Argument `apiKey` je vyžadovaný pre autorizáciu požiadavku príslušným developerom – teda developerským kľúčom. Posledný argument je `includeNutrition`, ktorý rozhoduje o tom, či má odpoveď zo serveru zahrnúť aj nutričné hodnoty receptu. Odpoveď s nutričnými hodnotami má podstatne viac bytov. Za parametrami nasleduje návratový typ funkcie. Funkcia vráti objekt `Recipe` zabalený v triede `Response` a ten zabalený v triede `Single`, vďaka čomu môže požiadavok prebehnúť asynchrónne.

```
interface RestApi {
        ...
    @GET("recipes/{id}/information")
    fun getRecipeDetail(
        @Path(value = "id", encoded = true) recipeId: String,
        @Query("apiKey") apiKey: String,
        @Query("includeNutrition") includeNutrition: Boolean? = true
    ): Single<Response<Recipe>>
        ...
}
```

Výpis 6.2: Metóda pre vyčítanie detailu recepta

Komunikácia s Firebase Realtime Database

Komunikácia s realtime databázou od Firebase prebieha iným spôsobom a nie sú potrebné žiadne knižnice. Kód z ukážky 6.3 sa vykoná v momente kedy užívateľ vytvorí nový vlastný recept a ten sa zálohuje na server. Disposable slúži pre väčšiu kontrolu nad asynchrónnymi volaniami, o ktoré momentálne ide. V prvom rade sa z lokálnej databázy aplikácie vyčítajú všetky užívateľom vytvorené recepty. Následne sa zo shared preferences získa univerzálny identifikátor prihláseného užívateľa. Potom príde na radu komunikácia s objektom `database` triedy `DatabaseReference`, odkazujúca na databázu na serveri. Volaním funkcie `.child(value)` sa referencia viac zanoruje. Až sa nastaví požadovaná cesta pomocou `userId` a definovanými konštantami, recepty sa uložia do vzdialenej Firebase databázy volaním `.setValue(it)`.

[7]Retrofit https://github.com/square/retrofit
[8]Moshi https://github.com/square/moshi

```
override fun updateCustomRecipes() {
    disposable?.dispose()
    disposable?.add(
        db.customRecipesDao().getAllRecipesSingle()
            .subscribe({
                val userUid = prefs.getUserUidFromSharedPreferences()
                var database: DatabaseReference = Firebase.database.reference

                database
                    .child("users")
                    .child(userUid)
                    .child("customRecipes")
                    .setValue(it)
            }, {
                Timber.e("updateCustomRecipes err: $it")
            })
    )
}
```

Výpis 6.3: Uloženie vytvorených receptov na server

6.5 Grafické rozhranie

Implementovanie grafického rozhrania v Androide sa realizuje tvorbou `XML` layoutov (rozložení). Programátor využíva štandardné UI elementy ako napríklad `Button` (tlačidlo), `Text View` (text), `Image View` (obrázok), prípadne iné elementy z použitých knižníc – viz. `Range Slider` v ukážke 6.4, ktorý je dostupný vďaka knižnici Material Design[9]. Najčastejšie som používal `Constraint Layout`, ktorá umožňuje pripnúť hrany elementu ku hranám iného elementu či prípadne ku hranám obrazovky. Okrem toho je možné nastaviť aj zviazanie, bariérovanie a iné nápomocné parametre pre definovanie UI.

V ukážke kódu 6.4 je definovaný `Range Slider` v rámci rodičovskej `Constraint Layout`, ktorý je použitý pre nastavanie kalorických limitov pri fitlrovaní vyhľadávania. Výsledné UI je zobrazené na obrázku 6.3. `Range Slider` s identifikátorom `rangeCalories` má nastavenú šírku cez celú obrazovku a zachovanú východziu výšku komponenty.

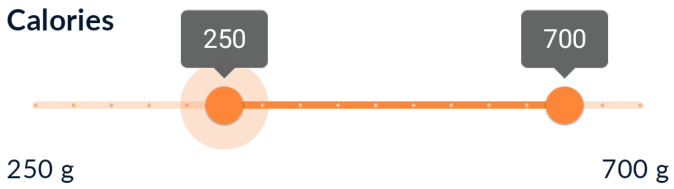
Veľkosť kroku `slider_values_calories_step` je 50, hodnotu získava z referenčného súboru `integers.xml`. Minimálna hodnota `slider_values_calories_min` je 0, maximálna hodnota je 800, udáva ju parameter `slider_values_calories_max`. Následne parameter `layout_constraintTop_toBottomOf` znamená, že horná strana elementu `rangeCalories` bude začínať pod spodnou hranou elementu `labelCalories`. Parameter `layout_marginTop` tým pádom nastavuje vzdialenosť od `labelCalories` na 12dp.

```
<com.google.android.material.slider.RangeSlider
    android:id="@+id/rangeCalories"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="12dp"
    android:stepSize="@integer/slider_values_calories_step"
```

[9]Použitie Material Design knižnice https://material.io/develop/android/docs/getting-started

```
    android:valueFrom="@integer/slider_values_calories_min"
    android:valueTo="@integer/slider_values_calories_max"
    app:layout_constraintTop_toBottomOf="@id/labelCalories"
    app:values="@array/slider_values_calories" >
</com.google.android.material.slider.RangeSlider>
```

Výpis 6.4: XML kód definujúci Range Slider pre nastavenie kalorických limitov



Obrázek 6.3: Výsledné zobrazenie `Range Slider` z ukážky kódu 6.4

Ďalšou veľmi využívanou komponentou v rámci implementácie užívateľského rozhrania bol `Recycler View`, ktorý slúži pre zobrazenie zoznamu položiek rovnakého typu – napríklad zoznam vyhľadaných receptov. `Recycler View` pri scrollovaní obsahu doslova recykluje predchádzajúce položky a uvoľnuje pamäť pre tie nasledujúce. Automaticky pracuje len s tými položkami, ktoré sú momentálne zobrazované. Pri implementovaní UI som taktiež využil referenčné súbory `strings.xml` pre definovanie všetkých statických textov, `colors.xml` pre definovanie hexadecimálnych kódov farieb, `dimens.xml` pre zavedenie často použitých rozmerov a `styles.xml`, v ktorom sú definované základné štýly tlačidiel, textov a iných dizajnových komponént pre dosiahnutie konzistentného užívateľského rozhrania.

6.6 Rozpoznávanie reči

Jedným z hlavných požiadavkov na aplikáciu je spracovanie hlasového vstupu používateľa. Jedná sa o stále viac populárnejšiu funkcionalitu s názvom `Speech to text`, ktorá konvertuje hovorené slovo do textových reťazcov. V Android SDK pre to existuje natívne API `Speech Recognizer`. V ukážke kódu 6.5 je metóda, ktorá spustí rozpoznávanie hlasového vstupu. Aplikácia v tomto momente ide do pozadia a na obrazovke sa zobrazí natívny dialóg pre odpočúvanie.

```
fun speechToText() {
    // Intent objekt pre rozpoznavanie hlasu
    val intent = Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH)

    // Nastavenie rozpoznavania reci vo volnej forme
    intent.putExtra(
       RecognizerIntent.EXTRA_LANGUAGE_MODEL,
       RecognizerIntent.LANGUAGE_MODEL_FREE_FORM
    )

    // Prednastavenie jazyka a titulneho textu v dialogu
    intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE, Locale.ENGLISH)
```

```
    intent.putExtra(RecognizerIntent.EXTRA_PROMPT, "I'm listening")

    try {
        // Spustenie intent akcie s vyziadanym kodom
        startActivityForResult(intent, REQUEST_CODE_STT)
    } catch (e: ActivityNotFoundException) {
        showErrorSnackbar(e.message)
    }
}
```

Výpis 6.5: Zapnutie rozpoznávania hovoreného slova

Následne po dohovorení prejde aplikácia opäť do popredia, čo spustí volanie metódy `onActivityResult()`. V ukážke kódu 6.6 je vidno, akým spôsobom sa získava výsledný reťazec z argumentu `data` funkcie. Ten je napokon vložený do vstupného textového poľa `queryInput`, čo spustí vyhľadávanie receptov.

```
override fun onActivityResult(
    requestCode: Int,
    resultCode: Int,
    data: Intent?
) {
                ...
    // Ziskanie dat z argumentov
    val result = data.getStringArrayListExtra(
        RecognizerIntent.EXTRA_RESULTS
    )

    if (!result.isNullOrEmpty()) {
        val recognizedText = result[0]
        binding?.queryInput?.setInputText(recognizedText)
    }
}
```

Výpis 6.6: Získanie textových reťazcov z hlasového vstupu

6.7 Rozpoznávanie ingrediencií z obrazu

Proces rozpoznávania ingrediencií z obrazu začína spustením fotoaparátu v rámci aplikácie vďaka natívnej knižnici **CameraX**[10]. Po stlačení spúšte sa fotka uloží do dočasnej pamäte a odošle sa na Firebase cloud pre spracovanie a rozpoznanie objektov z obrázku. V ukážke kódu 6.7 je viditeľná funkcia, ktorá sa stará o komunikáciu s cloudom v rámci image labelingu. Po získaní výsledkov sa používateľovi zobrazí bottom sheet s nájdenými objektami. Používateľ musí zaškrtnúť vyhovejúce ingrediencie. Slúži to ako spätná väzba a prevencia pred pridaním nesprávnych surovín. Odpoveď z cloudu obsahuje list objektov triedy `FirebaseVisionImageLabel`, ktorý obsahuje parametre:

- **text -** rozpoznaný štítok (reťazec/objekt)

[10] Android CameraX https://developer.android.com/training/camerax

- **entityId -** označenie entity v rámci Knowledge Graph
- **confidence -** dôveryhodnosť výsledku

```
override fun firebaseImageLabelingOnCloud(
    imageUri: Uri?,
    confidenceThreshold: Float
) {
    // Konvertovanie URI to bitmap formatu
    val bitmap = appContext.uriToBitmap(imageUri)

    // Ziskanie FirebaseVisionImage z bitmapu
    val image = FirebaseVisionImage.fromBitmap(bitmap)

    // Nastavenie rozpoznania z cloudu a pozadovanej doveryhodnosti
    val options = FirebaseVisionCloudImageLabelerOptions.Builder()
        .setConfidenceThreshold(confidenceThreshold)
        .build()

    // Vykonanie image labelingu s danymi nastaveniami
    val labeler = FirebaseVision.getInstance().getCloudImageLabeler(options)
    labeler.processImage(image)
        .addOnSuccessListener {
            // Ulozenie vysledkov
            imageLabelingState.postValue(
                Resource.success(it.toImageLabelingResponse())
            )
        }
        .addOnFailureListener {
            // Zachytenie chyby
            imageLabelingState.postValue(
                Resource.error(ErrorIdentification.Unknown(it.message))
            )
        }
}
```

Výpis 6.7: Funkcia, ktorá sa stará o image labeling vstupného obrázku

Kapitola 7

Testovanie a publikovanie

V tejto kapitole sa budem venovať priebehu testovania počas vývoja a finálneho publikovania mobilnej aplikácie medzi širokú verejnosť. Testovanie je neodmysliteľnou súčasťou procesu tvorby softwarového produktu a jeho opomenutie či podcenenie môže mať vážne následky. Existujú rôzne typy testovania, od automatizovaných testov až po užívateľské testovanie, ktoré môže reflektovať psychologický dopad aplikácie na koncového užívateľa. Testovanie pomáha nie len odhaliť chyby a nedostatky produktu, ale taktiež vypovedá o dosiahnutí stanovených požiadavkov, či prináša nápady na nové funkcionality a vylepšenia.

7.1 Testovanie

Testovanie užívateľského rozhrania

Pri tvorbe užívateľského rozhrania 4 prebehlo niekoľko iteratívnych testov mierených na potencionálnych používateľov aplikácie. Jednotlivé fázy návrhu som testoval na troch ľudoch a získal som tak spätnú väzbu, ktorá poukázala na správne respektíve nesprávne riešenia, ktoré boli v rámci ďalšej fázy návrhu zvalidované a prípadne poupravené.

Testovanie rozpoznávania ingrediencií z obrazu

Proces som testoval vo finálnej aplikácií a na vzorke dvadsiatich ingrediencií. Výsledky testovania sú uvedené v tabuľke 7.1. Jednotlivé ingrediencie som fotil oddelene. Výsledkom rozpoznávania z obrazu je pole reťazcov, odpovedajúce rozpoznaným objektom z pôvodného snímku. Dvanásť vzoriek bolo úspešne rozpoznaných, nakoľko sa názov fotenej ingrediencie nachádzal vo výslednom poli reťazcov. Z toho vyplýva **úspešnosť rozpoznávania 60%**. Pri zvyšných neúspešných pokusoch bolo potrebné, aby som názornú ingredienciu zadal do aplikácie iným spôsobom.

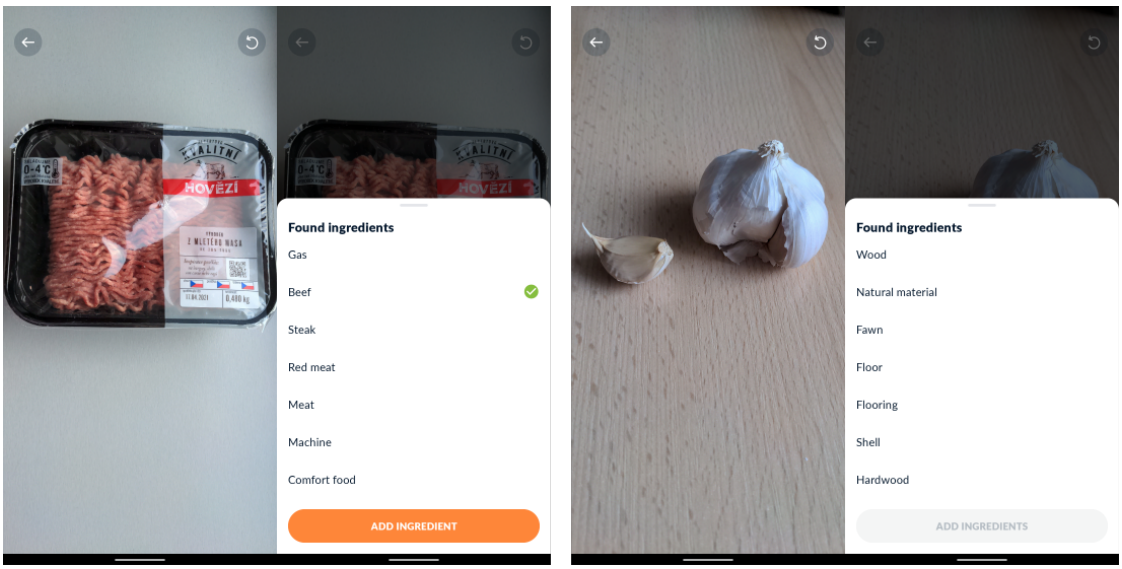
Pri rozpoznávaní objektov z obrázku služba vracia všetky nájdené objekty a nie len potraviny respektíve jedlo. Ukážku z testovania je možné vidieť na obrázku 7.1. Aplikácia rozpoznala, že sa jedná o hovädzie mäso, aj keď bolo v originálnom balení. Naopak nerozoznala cesnak a nepomohlo ani oddelenie jedného strúčiku. Pri detekcií sa často vyskytujú všeobecné termíny (food, vegetable, meat) a aj názvy pozadí, na ktorých sú ingrediencie položené (wood, floor). Pri implementácií rozpoznávania ingrediencií z obrazu 6.7, som hľadal spôsob akým by som obmedzil rozpoznávanie objektov iba na konkrétne kategórie `Food` a `Drink`, čím by sa eliminovalo rozpoznanie nepotrebných štítkov z ostatných kategórií. Žiaľ po naštudovaní dokumentácie a iných prameňov som zistil, že Firebase ML

Ingrediencia	Výsledok	Ingrediencia	Výsledok
Jablko	true	Káva	true
Citrón	true	Biele víno (v pohári)	true
Hovädzie mäso (balené)	true	Cheddar	false
Kuracie mäso	true	Cesnak	false
Cherry paradajka	true	Špenát	false
Žltá paprika	true	Vajce	false
Rohlík	true	Klobása	false
Chlieb	true	Arašidové maslo	false
Cibuľa	true	Zemiak	false
Kešu orechy	true	Mlieko (vo fľaši)	false

Tabulka 7.1: Testované ingrediencie

Kit túto funkcionalitu selekcie zatiaľ neumožňuje. V kapitole 7.1 som uvádzal, že každý rozpoznaný objekt obsahuje parameter s názvom `entityId`. Napadlo mi teda, že by som si selekciu kategórií mohol vykonávať v aplikácii, na základe tohto parametru. Napokon to však nebolo možné, pretože sa jedná o dve rozdielne veci, ktoré spolu nesúvisejú.

Je teda nutné uznať, že úspešnosť rozpoznávania potravín nie je príliš vysoká a ešte nejaký čas potrvá, kým bude realita porovnateľná s ideálnym scénarom. U niektorých ingrediencií je taktiež problematický obal, ktorý rozpoznávanie pochopiteľne zavádza a komplikuje. Pri všedných ingredienciach je možné využiť zadávanie ingrediencií pomocou rozpoznávania z obrazu, ale nedá sa to spoliehať. Aplikácia je pre tieto prípady dizajnovaná a disponuje ďalšími dvoma možnosťami ako zadať ingredienciu – textový alebo hlasový vstup.



Obrázek 7.1: Ukážka testovania rozpoznávania ingrediencií z obrazu

Testovanie finálnej aplikácie

Aplikáciu som testoval priebežne popri implementácií, takže väčšinu chýb som odhalil pri tvorbe samotných procesov a funkcionalít. Finálne riešenie som však dal otestovať štyrom testerom formou užívateľského testovania. To znamená, že som im zadal základné úkony, ktoré mali v aplikácií dosiahnuť a sledoval som pri tom ich správanie a schopnosť splniť úlohu. Testeri boli rôzneho veku, dvaja z nich mali pokročilé skúsenosti s aplikáciami a v oblaste IT, zvyšní dvaja len základné znalosti z bežného používania smartphonu. Pre testovacie účely som využil nástroj Firebase App Distribution[1], pomocou ktorého som zverejnil priebežné verzie aplikácie. Medzi pozorované úkony patrili:

1. Vyhľadaj svoj obľúbený recept
2. Nastav si intolerancie
3. Nastav si jedálniček na štvrtok v rámci tohto týždňa
4. Vytvor vlastný recept
5. Vyhľadaj recepty podľa dostupných surovín
6. Vyhľadaj recepty podľa dostupných surovín s využitím rozpoznávania obrazu

Najväčším problémom bol jednoznačne 3. bod. Testeri mali problém pri pridaní receptu do jedálničku. Najviac ich miatlo, že pri kliknutí na konkrétny obľúbený respektíve vyhľadaný recept, sa otvoril detail receptu miesto automatického pridania k pôvodne zvolenému dňu a času. Všeobecne bolo pochválené UI, farby a celkový dojem z výzoru aplikácie. Ďalej až traja testetri by taktiež očakávali možnosť pridania fotky pri tvorbe vlastného receptu. Nad touto funkcionalitou som pri návrhu rozmýšlal, ale rozhodol som sa ju z časového hľadiska v prvej verzie aplikácie vynechať. Testeri taktiež očakávali, že jedálniček bude umožňovať prepínať týždne, aby sa mohli inšpirovať predchádzajúcimi jedálničkami. Jeden tester by očakával možnosť vytvorenia vlastných kolekcií pre ukladanie receptov, ako napríklad *diétne jedlá*, *bezmäsité jedlá* a iné. Posledným poznatkom z testovania bol zvedavý požiadavok na tmavý režim aplikácie. V prvej verzii je dostupnosť *dark mode* vynechaná a nebol zohladnený pri návrhu užívateľského rozhrania.

Ku ďalším funkcionalitám by som ešte doplnil vlastné nápady, niektoré komplexnejšie budú spomenuté v kapitole 8. K jedálničku by som doimplementoval notifikácie, ktoré by sa vyskytovali podľa prednastaveného času v nastaveniach, napríklad večera o 19:00. Ku jednotlivým dňom jedálnička by som očakával spodnú sumarizačnú lištu, ktorá by počítala celkové kalórie a prípadne iné nutričné hodnoty. Na záver, domovská obrazovka by mala obsahovať *swipe to refresh layout* [2], ktorá by vykonala aktualizáciu doporučených receptov.

[1] Firebase App Distribution https://firebase.google.com/docs/app-distribution

[2] Swipe to refresh layout https://material.io/design/platform-guidance/android-swipe-to-refresh.html#usage

7.2 Publikovanie

Po fáze testovania chýba už len posledný krok – publikovať aplikáciu v obchode Google Play a zverejniť ju širokej verejnosti. Samotné vydávanie aplikácie je napokon komplikovanejšie ako by sa mohlo na prvý pohľad zdať. Rozhranie konzoly pre vývojárov pôsobí mierne zložito a Google ho často redizajnuje a sprísňuje aj samotný schvalovací proces. Pre vydanie aplikácie je potrebné vykonať nasledovné kroky:

- **Developerský účet -** je potrebné si vytvoriť vývojársky účet, za ktorý sa platí jednorázovo 25$. Tento krok som preskočil nakoľko vývojársky účet už mám
- **Záznam v obchode -** po prihlásení sa do účtu vývojára je nutné registrovať novú aplikáciu a následne vyplniť niekoľko potrebných údajov a formulárov ako sú napríklad názov aplikácie, krátky a dlhý popis, ikona, banner, snímky obrazovky, zásady ochrany osobných údajov, cieľová skupina, výskyt reklám a vekovo obmedzeného či nevhodného obsahu a iné
- **Schválenie aplikácie -** po vyplnení záznamu v obchode je možné nahrať *release* verziu aplikácie, ktorá sa odošle na schválenie firme Google. Schvalovací proces obvykle trvá 2 až 3 dni a donedávna sa jednalo len o automatizovaný proces. Dnes už sú aplikácie kontrolované samotnými zamestnancami a v prípade neschválenia vám príde rozsiahla správa špecifikujúca konkrétne nevyhovejúce podmienky

V prípade aplikácie Recipio schvalovací proces trval necelé dva dni a prebehol bez problémov. Po schválení aplikácie ma Google informoval emailom a od daného momentu bola aplikácia verejne dostupná na Google Play[3]. Avšak problémové je vyhľadanie aplikácie v prvých dňoch od vydania. Sámotný Google uvádza [1], že je potrebný istý čas pre správne zaindexovanie novovystavenej aplikácie v rámci obchodu.

[3] Aplikácia Recipio dostupná v obchode Google Play https://play.google.com/store/apps/details?id=com.lonchi.andrej.lonchi_bakalarka

Kapitola 8

Záver

Cieľom tejto práce bolo vytvoriť mobilnú aplukáciu pre spravovanie a vyhľadávanie receptov pre mobilné zariadenia s operačným systémom Android. Po definovaní hlavnej myšlienky a žiadaných funkcionalít som prezkúmal existujúce riešenia a analyzoval tak konkurenčné aplikácie. Zistil som, ktoré veci by sa dali spraviť lepšie a taktiež mi to ukázalo niekoľko fungujúcich riešení problémov, ktorými som sa inšpiroval a nemusel som tak *objavovať koleso*. Následne som vytvoril prototypy užívateľského rozhrania, ktoré som iteratívne podroboval testovaniu, získaval spätnú väzbu od potencionálnych používateľov a dopracoval sa tak ku finálnemu grafickému návrhu. Tvorba UI bola do istej miery ovplyvňovaná aj so súčasne prebiehajúcim výberom technológií a analýze možných riešení. A to najmä pri procese rozpoznávania ingrediencií z obrazu či štruktúre dát receptov od služby Spoonacular Food API. Pri implementácií som narazil na niekoľko menších problémov, kvôli ktorým som musel spätne zasahovať do grafického návrhu, napr. pri tvorbe vlastného receptu resp. zadávaniu surovín, nemám odkiaľ získať obrázok danej ingrediencie. Testovanie poukázalo na pár nedokonalostí v rámci finálneho riešenia, ale taktiež otvorilo nové možnosti a pohľady na nové funkcionality pre budúci rozvoj aplikácie.

Výsledkom je plne funkčná mobilná aplikácia, ktorá spĺňuje všetky počiatočné požiadavky. Pri vývoji boli dodržané odporúčané štandardy a jej užívateľské rozhranie je navrhnuté podľa moderných trendov. Finálny produkt – aplikácia Recipio – je dostupná širokej verejnosti a jej hodnotenie a spätnú väzba od reálnych používateľov je možné vidieť v zázname obchodu Google Play[1]. Samozrejme si uvedomujem obrovskú konkurenciu, ktorej je aplikácia vystavená. Napriek tomu sa jedná o plne funkčné riešenie, na ktorom sa dá ďalej stavať a pridávať nové zaujmavé funkcionality, ktoré aplikáciu opäť posunú o krok vpred. Vzhľadom k rýchlosti vývoja mobilných aplikácií existuje pomerne malé množstvo tlačenej literatúry a preto boli použité predovšetkým digitálne dostupné zdroje.

V budúcnosti by som sa samozrejme chcel venovať oprave prípadných chýb a implementovaní nových funkcionalít. Napríklad pridanie `OCR`, optické rozpoznávanie znakov, ktoré by z pôvodných kuchárok či digitálnych poznámok automaticky vytvoril recept požadovaného formátu. Menej náročnou funkcionalitou je pridanie exportovania receptu z aplikácie do formátu `pdf`. Vzhľadom na vzrastajúci trend – *dark mode* – by aplikácia mala disponovať aj tmavým režimom. Pevne verím, že ďalšie nápady na zlepšenie navrhnú aj samotní používatelia aplikácie formou emailu alebo zanechaním komentáru v obchode Google Play.

[1] Aplikácia Recipio dostupná v obchode Google Play https://play.google.com/store/apps/details?id=com.lonchi.andrej.lonchi_bakalarka

Literatura

[1] *App visibility and discovery issues* [online]. Google Play Console Support [cit. 2021-04-30]. Dostupné z: https://support.google.com/googleplay/android-developer/answer/9042516?hl=en.

[2] *Mobile Operating System Market Share Worldwide* [online]. Glob Stats [cit. 2021-02-13]. Dostupné z: https://gs.statcounter.com/os-market-share/mobile/worldwide.

[3] *Platform Architecture* [online]. Android Developers [cit. 2021-02-22]. Dostupné z: https://developer.android.com/guide/platform.

[4] *Save data in a local database using Room* [online]. Android Developers [cit. 2021-04-19]. Dostupné z: https://developer.android.com/training/data-storage/room.

[5] *The 'Paleo Diet' – Back to the Stone Age?* [online]. Harvard Health Publishing, červenec 2015 [cit. 2021-03-01]. Dostupné z: https://www.health.harvard.edu/diet-and-weight-loss/the-paleo-diet-back-to-the-stone-age.

[6] *Android vs iOS: Which Platform to Build Your App for First?* únor 2018 [cit. 2021-02-20]. Dostupné z: https://medium.com/@the_manifest/android-vs-ios-which-platform-to-build-your-app-for-first-22ea8996abe1#:~:text=It%27s%20faster%2C%20easier%2C%20and%20cheaper,Swift%2C%20Apple%27s%20official%20programming%20language.

[7] *Use Android Jetpack to Accelerate Your App Development* [online]. Android Developers, květen 2018 [cit. 2021-02-11]. Dostupné z: https://android-developers.googleblog.com/2018/05/use-android-jetpack-to-accelerate-your.html.

[8] *A User Interface is Like a Joke* [online]. UX World, červen 2018 [cit. 2021-02-23]. Dostupné z: https://uxdworld.com/2018/06/16/a-user-interface-is-like-a-joke/.

[9] Chatterjee, A. *Null Safety in Kotlin* [online]. The Startup, červen 2020 [cit. 2021-02-13]. Dostupné z: https://medium.com/swlh/null-safety-in-kotlin-88298e64a1dc.

[10] Hongbeomi. *Create Android app with MVVM pattern simply using Android Architecture Component* [online], 15. dubna 2020 [cit. 2021-04-13]. Dostupné z: https://medium.com/hongbeomi-dev/create-android-app-with-mvvm-pattern-simply-using-android-architecture-component-529d983eaabe.

[11] Lang, D. *Android Activity Life Cycle* [online]. 2017 [cit. 2021-02-11]. Dostupné z: http://landenlabs.com/android/info/activity-life-cycle/activity-life-cycle.html.

[12] Wu, X. *Recent advances in deep learning for object detection* [online]. Science Direct, 5. července 2020 [cit. 2021-04-26]. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0925231220301430?via%3Dihub.

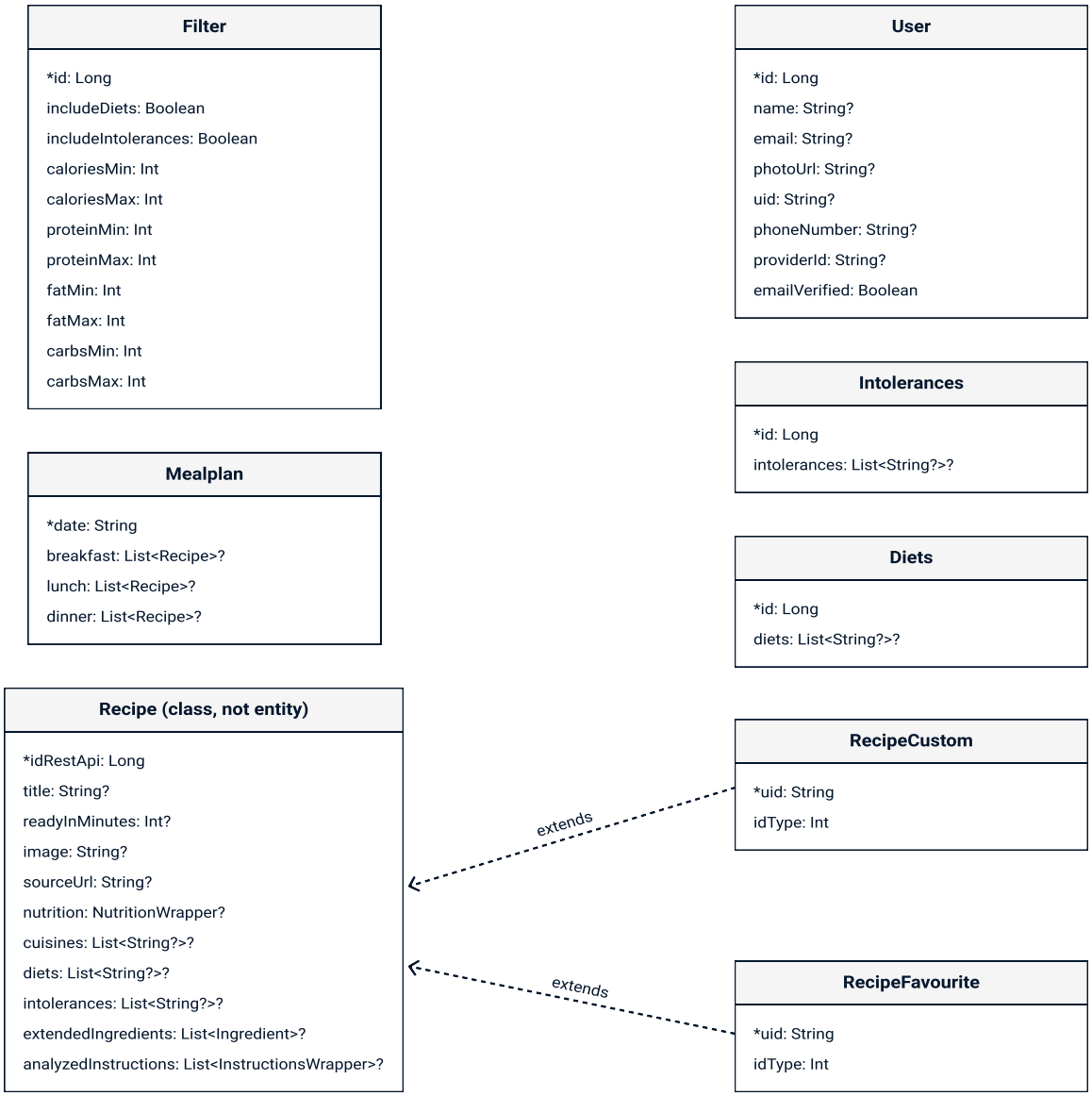
Příloha A

Obsah priloženého média

- **src_android/** zdrojové súbory aplikácie
- **src_latex/** zdrojové súbory dokumentácie vo formáte LaTeX
- **LICENSE.txt** licencia
- **README.txt** informácie
- **recipio_1.0_release.apk** inštalačný súbor aplikácie
- **xlonci00.pdf** táto práca vo formáte PDF

Příloha B

Entitný diagram



Obrázek B.1: Štruktúra použitých entít

Příloha C

Plagát

Obrázek C.1: Prezentačný plagát k aplikácií